

LINKÖPING UNIVERSITET

TNM094

PROJEKTRAPPORT

TEAM VIRTUELLA VATTENFÄRGER

Tim BRODIN
Alexander CEDERBLAD
Kristina ENGSTRÖM
Sofia WENNSTRÖM
Anton ÖSTERBLAD

timbr473
alece341
krien026
sofwe496
antos600

KUND: Anna ÖST

HANLEDARE: Karljohan LUNDIN PALMERIUS

3 juni 2014

Sammanfattning

Rapporten beskriver arbetsprocessen för ett projekt för kandidatexamen (kurskod TNM094) vid Linköpings Universitet. Studenterna har under kursen fått tillämpa agil utvecklingsmetodik vilket, tillsammans med ett flertal reflektioner kring systemutveckling, redogörs i rapporten. Vidare diskuteras vilka verktyg som utnyttjats för åstadkommet resultat.

I projektet vidareutvecklas en utställningsstation på Visualiseringsscenter C för realtidsimulering av vattenfärger. Detta genom att skapa möjligheten att kunna spara en målad bild och publicera den på nätet. En webbapplikation skapas, där både galleri och statistik av de bilder som tillåts publicering visas. Användare som väljer att spara sin bild kan namnge konstnär och konstverk, dela bilden på Facebook samt tillåta att bilden visas publikt på hemsidan (<http://vis-c.se>). Administratörer har genom webbapplikationen möjligheten att administrera konstverk.

Rapporten diskuterar verktyg som används för att underlätta utvecklingen samt de tillsatta rutiner som samtliga gruppmedlemmar ska följa.

Innehåll

Sammanfattning	i
1 Inledning	1
1.1 Bakgrund	1
1.2 Syfte och frågeställning	1
1.3 Avgränsningar	2
1.4 Metod	2
1.5 Typografiska konventioner	2
2 Arbetsprocess	4
2.1 Uppstart	4
2.2 Planering	4
2.3 Möten	4
2.4 Verktyg	4
2.5 Systemarkitektur	5
2.6 Testning	6
2.6.1 Användartest	6
2.6.2 Enhetstest	6
2.6.3 Implementationstest	6
2.7 Kundkontakt	6
2.7.1 Planering och kontakt	6
2.7.2 Möten	6
2.8 Kravhantering	7
2.9 Examinatorskontakt	7
2.10 Sprints och Stories	7
2.11 Rutiner	8
3 Implementering	9
3.1 Vattenfärgssimuleringen	9
3.1.1 Gränssnittet	10
3.2 RESTful API	10
3.3 Webbapplikationen	11
3.3.1 Model-View-Controller	11
3.3.2 Dependency Injection	11
3.3.3 Offentlig del	11
3.3.4 Administrativ del	12
3.3.5 Distribuera	12
3.3.6 Galleri	12
3.3.7 Statistik	13
3.3.8 Databashantering	13
3.3.9 Dokumentation	14

4 Resultat	15
4.1 Vattenfärgssimuleringen	15
4.2 Integration mellan station och webbapplikation	16
4.3 Webbapplikationen	17
5 Diskussion och framtida arbete	21
6 Slutsatser	23
A Individuell prestation	25
A.1 Tim Brodin	25
A.2 Alexander Cederblad	25
A.3 Kristina Engström	25
A.4 Sofia Wennström	25
A.5 Anton Österblad	25
B Projektplan	27
B.1 Kort beskrivning	27
B.2 Teknisk beskrivning	27
B.3 Infrastruktur och systemutveckling	28
B.3.1 Utställningsmonter	28
B.3.2 Bildpublicering och inspektion	28
B.3.3 <i>Stories</i> och <i>tasks</i>	29
B.3.4 Versionshantering och kodkvalité	30
B.3.5 Kodgranskning	30
B.3.6 Projekthantering	30
B.3.7 Kodstandarder	30
B.3.8 Kravhantering	31
B.3.9 Modelleringsstandard	31
B.3.10 Tidsplan	32
C Sprints	33
C.1 Sprint 1: Stationen	33
C.2 Sprint 2: Webbapplikationen	34
C.3 Sprint 3: Administration	34
C.4 Sprint 4: Sista sprinten	36
D Backlog	37
E RESTful API dokumentation	39

Kapitel 1

Inledning

I kursen TNM094, Medietekniskt kandidatprojekt, ska studenterna på medieteknikprogrammet i Norrköping tillämpa agil utvecklingsmetodik för att lösa ett eller flera problem inom systemutveckling åt kund. Denna rapport beskriver arbetsprocessen och resultatet för ett projekt som genomförts i kursen.

1.1 Bakgrund

På Visualiseringscenter C i Norrköping fanns en station som gav besökare möjligheten att måla med virtuella vattenfärgar. Stationen är ett tidigare examensarbete gjort av två medieteknikstudenter [1] som involverade fluidsimulering samt ett tredimensionellt trackingsystem med magnetfält.

Utställningen fick bra respons på centret och projektgruppen fick i uppdrag att vidareutveckla stationen för att skapa mervärde för användarna. Kunden bad om funktionalitet för att spara de konstverk som gäster på visualiseringscentret målat. Publicering ska ske till Internet, för intern visning på centret i form av bildspel eller liknande. En administrationsdel ska dessutom finnas för möjligheten att administrera de bilder som publicerats för visning på hemsidan och centret.

1.2 Syfte och frågeställning

Projektet är menat att lösa de problem och önskemål som kund på Visualiceringscenter C framställt. Den efterfrågan med högst prioritet är den redan nämnda publiceringsplattformen med administrationsmöjligheter. En annan efterfrågan är möjligheten att föra statistik över användandet av utställningen tillsammans med att visa upp denna statistik, förslagsvis på en hemsida. Samtliga uppdrag är utformade efter följande frågeställning:

- Kan utökad funktionalitet göra stationen mer attraktiv?
- Hur kan stationen anpassas efter dess avsedda målgrupp?
- Hur ska publiceringsplattform och målarstation samverka?
- Vilka aspekter måste tas hänsyn till då stationen nyttjas för publikt bruk?
- Vad har stationens redan befintliga kod för utvecklings- och förbättringspotential?
- Vad för verktyg och metoder kan användas för att underlätta och förbättra utvecklandet av mjukvara i grupp?

- På vilket sätt underlättas systemutveckling av testning samt rutiner för kodgranskning och publivering?
- Hur och när kan designmönster användas för att skapa exempelvis moduläritet i systemet samt underlätta testning?

En stor del av projektet kommer involvera att utveckla metoder och rutiner för att arbeta i grupp med mjukvaruutveckling. Dessa rutiner kommer grunda sig i den agila utvecklingsmetoden *Scrum*.

1.3 Avgränsningar

På grund av tidsbrist samt faktumet att stationen avvecklas på centret prioriteras vissa saker bort. Exempelvis ett önskemål från kund att ge ökad precision av penseln.

Vid utveckling av mjukvaran för vattenfärgssimuleringen är gruppens arbetsmiljö begränsad till Windows operativsystem.

1.4 Metod

Vid agil utveckling är begreppet *Scrum* vanligt förekommande. *Scrum* är ett av flera agila tillvägagångssätt [2] och används till att arbetsfördela uppgifter i gruppen under projektets gång. *Scrum* består av *backlog*, korta dagliga möten och *sprints*. I *backlog* sätts alla krav upp och *sprints* är korta tidsperioder där uppsatta krav bearbetas. Team som arbetar enligt *Scrum* består av en *Scrum Master*, en produktägare samt ett utvecklingsteam [3].

Projektgruppen arbetar enligt *Scrum* med visionen att stämma träff med kund efter varje avslutad *sprint*. Varje *sprint* delas upp i mindre uppgifter, tasks, som fördelades jämnt mellan gruppens medlemmar.

Då projektgruppen endast består av fem personer ingår alla medlemmar i utvecklingsteamet, där alla har en biroll. Gruppen är indelade i följande biroller: Projektledare, *Scrum Master*, produktägare, ansvarig för kodgranskning, koordinator samt ansvarig för projektrapporten. För utförligare beskrivning av posterna, se bilaga A.

Inför varje *sprint* hålls ett möte för att planera sprinten samt strukturera det kommande arbetet. Alla medlemmar får en eller flera uppgifter som ska vara klara innan sprintens slut. Varje *sprint* består av *stories* som i sin tur är indelade i *tasks*. I slutet av varje *sprint* märks varje story som ”klar” eller inte.

Efter varje *sprint* diskuteras det arbete som utförts i två steg. En *sprint retrospective*, eller återblick, och en *sprint review*, eller granskning. I återblicken diskuterar utvecklingsgruppen tillsammans med *Scrum Master* om hur arbetsprocessen har gått och vad som kan förbättras till nästa *sprint*. Granskningen behandlar bland annat det arbete som både har och inte har slutförts.

1.5 Typografiska konventioner

Följande typografiska konventioner används i rapporten:

- *Kursiv* anger en engelsk term.

- **Kursiv och fetstil** avser presentation av programvara.

Kapitel 2

Arbetsprocess

2.1 Uppstart

Ingen i gruppen hade sedan tidigare testat på vattenfärgssimuleringen vid stationen och visste därmed inte heller vilka mål som vore rimliga för projektet. Målarstationen finns vid projektets start inte tillgänglig och tillhörande mjukvara saknas en kort tidsperiod. Gruppen befinner sig därmed i en knepig situation, inte minst med tanke på projektplanen som ska lämnas in i tidigt skede. Efter kontinuerlig e-mailkontakt samt ett möte med ytterligare personal på visualiseringsscentret reds dock läget ut och gruppen får allt materiel som behövs för att slutföra projektplanen i tid. Redan i denna fas utelämnas en hel del idéer som resonerats kring att vidareutveckla på grund av de begränsningar som systemet omfattas av.

2.2 Planering

Vid projektets första möten beslutas att arbetet ska fördelas över fyra *sprints*. Dessa *sprints* struktureras i sin tur noggrannare allt eftersom de bearbetas. Anledningen till detta är att det helt enkelt inte har fungerat att exakt förespå vad som kommer hinnas med under varje arbetsperiod. Veckan innan en *sprint* startar utformas stories och tasks som fördelas bland utvecklarna. Om en task inte hinnas med under en *sprint* flyttas den till nästa *sprint*. Mellan varje *sprint* hålls ett möte för att utvärdera arbetet och resultatet från föregående *sprint*. Utifrån detta planeras nästkommande *sprint*.

2.3 Möten

Varje arbetsdag startar med ett *Scrum*-möte. Där redogör gruppens medlemmar för vad som färdigställts och vad som ska bearbetas härnäst. På så sätt får utvecklarna god överblick för vad som pågår i projektet och varje gruppmedlem måste arbeta för att fullfölja planeringen.

2.4 Verktyg

Verktygen som används har stor påverkan på arbetsprocessen i detta projekt. De är avgörande för hur gruppen väljer att gå till väga för att lösa olika sorters problem, och dessutom hur problemen utformas.

Vid kravhanteringen utnyttjas det webbaserade vertyget *Agilefant*¹. Där sätts både *sprints* och stories upp tillsammans med tillhörande tasks. I Agilefant ges en tydlig överblick på arbetsgången där det är

¹<http://agilefant.com>

synligt vem i gruppen som gjort vad.

Vid design av webbapplikationens gränssnitt används **Adobe Photoshop** samt **Adobe Illustrator**. Dessa verktyg utnyttjas även för att förgylla målarvyn vid stationen.

I **Google Drive** bevarar och skriver gruppen alla sina dokument. Samtliga filer är åtkomliga för hela gruppen och ändringar delas smidigt med varandra.

Den slutgiltiga projektrapporten skrivas och kompileras med hjälp av **L^AT_EX**.

För versionshantering används **git** som är välanvänt mjukvara. Verktyget tillåter att göra ändringar i kodbasen utan att direkt behöva oroa sig för förlorad kod eller *backup*. Detta följer med gratis om git används på rätt sätt. Git underlättar för samarbete i grupp genom att i små steg utöka kodbasen på ett explicit sätt för att sedan sammanställa en gemensam mjukvara. För att enkelt dela kodbasen mellan gruppmedlemmarna används **GitHub**² för möjligheten att spara allt i molnet. GitHub underlättar ytterligare sammanställningen av mjukvaran med egna verktyg som “*pull request*” för att enkelt hålla reda på all kod samt diskutera problem och lösningar i mjukvaran.

När webbapplikationen utvecklas vill gruppen ha identiska utvecklingsmiljöer med produktionsservern. Detta är ett hinder då produktionsservern är ett webbhotell med endast FTP-åtkomst. En mapp på en delad server helt enkelt. För att återskapa denna begränsade miljö på ett enkelt sätt installerar var gruppmedlem en **AMP**-installation på sin dator, vilket innebär ett paket innehållande en Apache-server, en **MySQL**-databas samt **PHP** av relevanta versioner. Dessa installationer konfigureras på samma sätt på var medlems dator. Detta möjliggör testning och köring av sin kod lokalt.

För att underlätta kodgranskningsprocessen används **Travis-CI**. Travis är kopplat till webbapplikationens *GitHub-repository*. När ett anrop gällande tillägg av kod körs startas Travis automatisk som bygger och testar koden vilket underlättar “*pull request*”-arbetsprocessen. Det är en fördel att köra testen på en oberoende server för att garantera kodens kvalité och funktionalitet. Travis kan även meddela intressenter över *e-mail* när tester lyckas och misslyckas.

Pakethantering är ett sätt att hantera de tredjepartspaket som en applikation är beroende av. Den hjälper till att hantera versioner av tredjepartsmjukvara samt nedladdning och i vissa fall även användningen av mjukvaran. Det mest anammade verktyget för PHP heter Composer³ och löser elegant dessa problem. En strukturerad textfil underhålls med de behov som finns och ett kommandotolksprogram tillåter utvecklaren att installera och använda dessa på ett smidigt sätt. Pakethanterare finns för flera olika specifika programmeringsspråk såsom bower⁴ för JavaScript och pip⁵ för Python.

PHPUnit är det ramverk som används i projektet för enhets- och integrationstest av PHP.

2.5 Systemarkitektur

Eftersom mjukvara för utställningen redan existerade valde gruppen att hålla egen kod skilt från den ursprungliga i allra högsta grad. Detta gav upphov till utmaningen att skriva så modulärt som möjligt. Tilläggen i den ursprungliga kodbasen är endast en liten del av hela mjukvaran och är väl separerad

²<https://github.com>

³<https://getcomposer.org>

⁴<http://bower.io>

⁵<https://pip.readthedocs.org>

från varandra genom att fristående moduler skapas med minimala gränssnitt som kan användas i den körande koden.

2.6 Testning

Det finns tre olika typer av test som berör projektet: användartest, enhetstest och implementationstest.

2.6.1 Användartest

I projektets början då stationen fortfarande var igång erbjöds gruppen av kund möjligheten att utföra användartester under centrets öppettider. Detta hade varit ett ypperligt tillfälle att testa om användargränssnittet var tydligt och lättförståeligt för målgruppen som utvecklarna arbetat för. Tyvärr gavs aldrig denna möjligheten.

2.6.2 Enhetstest

För att kunna skriva kod i grupp är det viktigt att varje medlem skriver test för de moduler som skrivs, detta för att försäkra om att de fungerar på ett förväntat sätt. För att skriva enhetstest måste mjukvarans ansvar delas upp i mindre delar. Detta kan tolkas som ”alla klasser ska ha ett ansvar”. Även testskrivandet förenklas då det går att separera testen för att slippa behöva oroas för att de olika klassernas beroende ska gå sönder under körningen av testen. Gruppen strävar inte efter att ha 100% täckning av enhetstester, men att ha täckning av en majoritet av de moduler som skrivs till projektet.

2.6.3 Implementationstest

För att försäkra att exempelvis webbapplikationen fungerar som det är tänkt kan det vara fördelaktigt att skriva enhetstest för implementationerna i sin helhet. Detta för att slippa att manuellt klicka runt på exempelvis hemsidan för att garantera att inget fel uppstår eller att databasen beter sig som väntat.

2.7 Kundkontakt

För att sätta upp mål och kravspecifikationer för projektet krävs en bra kundkontakt. Detta har i projektet främst hanterats via möten och e-mail.

2.7.1 Planering och kontakt

Produktägaren i utvecklingsteamet sköter via e-mail kontakten med kund, men hela gruppen beslutar tillsammans med kund när ett möte är av behov. Möten äger rum då en *sprint* är avslutad eller då utvecklarna vill diskutera framtida arbete med kund. Både kundens önskemål och gruppens kompetens vägs samman vid beslut gällande vad som ska utvecklas i projektet.

2.7.2 Möten

Ett första möte med kund bokas tidigt i arbetsprocessen för att ge utvecklingsgruppen en tydlig bild av de förväntningar och önskemål som fanns att bearbeta. Bilden av projektet innan mötet var att endast lägga fokus på publicering av bilder, men efter att kunden ävenbett utvecklarna om eventuell förbättring i precisionen av målandet planeras arbetet annorlunda. Efter noggrann granskning av gruppens förmåga och ambitioner beslutas det att prioritera funktionen att spara bilder tillsammans med ett väl fungerande administrativt system. Att integrera ytterligare precision, exempelvis valmöjligheten att

måla med blyerts, anses för komplicerat inom de tidsramar som satts upp. Ett annat önskemål från kund var möjligheten att blockera obscena bilder.

Vid andra kundmötet redovisar gruppen projektplaneringen som godkänns av kund. Gruppen presenterar en idé gällande presentation av statistik som uppskattas av kund. Att utveckla applikationen till en möjlighet för spel känns dock överflödigt enligt kund, och utvecklingsteamet bestämmer sig för att sedermera inte bearbeta en lösning för det. Efter diskussion gällande publiceringsfunktionen beslutas att den mest optimala lösningen är att låta administratören kunna slå på och av ”direkt publicering”. Det innebär att administratören har möjlighet att avgöra om de bilder som målas bör ha tillgång till den tänkta hemsidans galleri utan att först godkännas eller inte. Det är dock svårt att måla obscen i och med hur precisionsfattig målarstationen är. Men kunden och utvecklingsteamet är ändå eniga om att någon typ av blockering är nödvändig att implementera.

Kunden har i senare möten inte mycket att anmärka på då gruppen visar upp vad de åstadkommit, men ger besked om att målarstationen ska tas bort från utställningen på visualiseringscentret. Det är dock inget som hindrar projektet att vidareutvecklas och kunden är nöjd med gruppens resultat.

2.8 Kravhantering

Kravlistan tas fram utifrån kundens önskemål och formuleras i projektets *backlog*. Exempel på hur gruppens *backlog* utformades finns att hitta i bilaga D. Utifrån önskemålen i backlogen utvecklas stories fram som beskriver en funktionalitet med väl specificerade krav. Dessa stories delas i sin tur upp i tasks som är de arbetsuppgifter som en gruppmedlem ska behandla under en kortare tid, en halv dag till två dagars arbete i gruppen. Backlog uppdateras under projektets gång där krav tillhörande funktioner som väljs bort att implementeras raderas och krav för nyfunna funktioner tillkommer.

2.9 Examinatorskontakt

Vid halvtidsseminariet i kurserna fanns ännu inte en tydlig systemarkitektur för projektet, men efter diskussion med examinatörn blev det klart att det kunde vara av behov. Projektet består av flera delmoment och en helhetsbild över hur dessa fungerar tillsammans utgör en väsentlig del för förståelse av projektet samt dess omfattning.

Övrig kontakt med examinatörn har skett via e-post.

2.10 Sprints och Stories

Arbetet planeras för och utförs under fyra *sprints*. Samtliga *sprints* planeras att pågå under 10 arbetsdagar. Nedan redogörs hur dessa *sprints* infattades vid projektets uppstart. För ytterligare specifikation gällande *sprints*, se bifogad bilaga C.

Under första sprinten ska interaktionen att spara bild färdigställas tillsammans med ett lättförståeligt grafiskt gränssnitt. I sprinten ingår även undersökningar av lagringsalternativ för bilder och hur trackingssystemet fungerar som interaktionsmedel.

I andra *sprinten* utformas ett annat gränssnitt, nämligen hemsidans. På hemsidan ska ”godkända” (administratörens beslut) verk finnas att beskåda, gilla och dela på sociala medier. Ett ytterligare mål

under denna *sprint* är att påbörja en lösning över hur statistik från målarstationen kan lagras och presenteras. Statistiken är gruppens egna förslag som ger användarna kunskap om vilken färg som är populärast, hur lång tid det i snitt tar för en bild att färdigställas etcetera.

I *sprint* tre behandlas administrationen av publika bilder. Administratören ska själv kunna bestämma huruvida bilder tillåts direkt publicering eller inte och så även sälla bort de bilder som anses olämpliga. Ett inloggningssystem till hemsidan ska dessutom implementeras.

Sprint fyra består av det utvecklarna benämner ”bonusfunktioner”, exempelvis ökad precision. Där ingår funktioner som prioriterats lägre, utan att för den sakens skull blivit bortglömda.

2.11 Rutiner

Den grundrutin som följs under projektarbetet när det gäller mjukvaruutvecklingen sker i dessa steg:

1. Ta ansvar för en task.
2. **Skriv kod.**
3. Se till att det finns test för implementationen om behov finns.
4. Se till att ha den senaste kodbasen tillsammans med tillägget som precis har implementerats.
5. Kontrollera att alla test körs korrekt.
6. Kontrollera att specificerad *style guide* följs.
7. Gör en Pull Request till projektets Github-repository.
8. Låt koden granskas av en annan gruppmedlem. Diskutera problem och läsbarhet. Gör tillägg och/eller ändringar. Repetera från **punkt 4**.
9. Låt en gruppmedlem lägga till implementationen i kodbasen.

Kapitel 3

Implementering

3.1 Vattenfärgssimuleringen

Att arbeta med befintlig kod innebär mycket stoff för utvecklarna att sätta sig in i. Kvalitén på den befintliga mjukvaran är bristande i många aspekter, vilket kan väntas av ett examensjobb som mjukvaran grundar sig från. Mjukvaran för vattenfärgssimuleringen är skriven i C++. Fluidsimuleringen sker huvudsakligen i OpenGL med shaders skrivna i GLSL, dessa ignoreras helt och hållt av projektgruppen. Den ursprungliga koden saknar strukturerade kommentarer. Ett försök att generera en översikt över systemet görs med **Doxxygen**¹, men utan resultat. Systemet importeras i **Astah**² och utifrån den informationen kunde ett klassdiagram skapas för att generera en överblick. Kodbasen består av en huvudsaklig fil som kör all kod tillsammans med vissa hjälpklasser samt ett skräddarsytt ramverk för 3D-trackingen. Dessa är ihopkopplade med globala variabler, vilket i sin tur skapar oorganiserad kod. Mjukvaran saknar test och många varningar uppstår vid kompilering. Gruppen får vetskapp om att stationen har upplevt vissa problem och att anställda på centret försöker åtgärda dessa. Utifrån denna lärdom bestämmer gruppen att ändra så lite som möjligt i kodbasen för att på så sätt undvika problem som kan uppstå med oorganiserad kod. De tillägg som behövs skrivs så modulärt som möjligt. Vidare bör det nämnas att det trackingsystem som stationen använder fortfarande är under utveckling och utvecklingsgruppen ombeds därför att i största möjliga mån undvika att lägga för stor vikt och tid på detta.

För att finna utvecklingsmöjligheter och diskutera idéer används *brainstorming* som metod. Ett antal funktioner samt lösningar förs på tal och för att välja vilka som ska prioriteras högst görs en “*Trade-off*”-analys. Detta innebär att de alternativa lösningarna som tagits fram vägs mot varandra för att sedan låta gruppen tillsammans avgöra vilka som är viktigast för produktens slutliga resultat.

Det viktigaste momentet som måste läggas till i mjukvaran för stationen är stödet för att spara en bild som en användare har målat. Denna funktionalitet kan implementeras på ett flertal vis med respektive för- och nackdelar. För flexibilitet sparar innehållet i molnet och, som nämnt tidigare, skrivs så lite kod som möjligt för stationens mjukvara samt att koden hålls separat. På ett naturligt sätt väljs ett gränssnitt mot en webbapplikation för att hantera bearbetning och lagring av bilder som sparas av användare. Alltså behöver endast möjligheten för stationen att kommunicera med Internet implementeras för att kunna skicka anrop mot detta gränssnitt. Möjligheten att kommunicera med Internet implementeras med hjälp av cURL³ (mer specifikt libcurl) som är ett bibliotek för kommunikation för en stor mängd protokoll för Internettrafik. En klass skapas runt den curl-funktionalitet som krävs för applikationen för att göra implementationen mer modulär och läsbar. Ett HTTP-anrop skickas alltså,

¹<http://www.stack.nl/dimitri/doxygen>

²<http://astah.net>

³<http://curl.haxx.se>

med tillhörande bilddata, till en webbserver och behandlas där. Ett svar på detta anrop tas emot av stationen som en bild på en textsträng i form av en webbadress eller felmeddelande som kan visas som en textur. Att skicka ett svar som en bild kan verka dumt, åtminstone att skicka relativt stora mediafiler, jämfört med en JSON-sträng över Internet. De uppenbara problemen med detta är förbrukningen av bandbredd samt hastighetsförlusten för att skicka större filer över Internet. Den största fördelen däremot är att stationens mjukvara inte behöver ha funktionalitet för att skapa och visa text, den behöver endast kunna visa en textur som redan existerar.

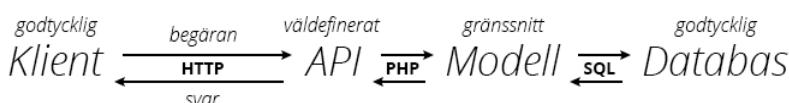
3.1.1 Gränssnittet

Stationen har två vyer, en som visar pappret samt en som visar en virtuell verklighet av simuleringen i tre dimensioner. Pappret som användaren målar på ska vara fritt från annan interaktion än att bara måla för att behålla ”måla-med-vattenfärg”-känslan, användaren ska alltså klara sig utan VR-vyn. Om en användare vill spara en bild måste därför exempelvis adressen till en bild som en användare sparar visas i VR-vyn. Denna vy måste tydligt visa interaktionsmöjligheterna för att rensa pappret och spara en bild. Alternativt skapas fysiska objekt för att antyda interaktion på liknande sätt som redan existerar med målarfärgerna. Det är viktigt för kunden att användaren aldrig ska lyckas interagera med dessa sekundära entiteter av misstag, för att behålla den redan nämnda känslan. Detta förhindras delvis genom att endast tillåta interaktion om interaktionen varar en viss tidsperiod, exempelvis att penseln hålls i ett bestämt område i en sekund för att exempelvis starta processen att spara en bild.

Att stationen brukas som publik utställning påverkar bearbetningen. Eftersom främst barn använder stationen är det viktigt att kontinuerligt fokusera på enkel kommunikation. Detta genom att exempelvis ersätta textinnehåll med ikoner som barn förstår. Estetiskt sett är VR-vyn enkel och tråkig, fördelen med det är att fokus läggs på att visa interaktionen och inte att det ska se snyggt ut. Trots detta ändras de flesta texturer i scenen mot bilder som reflekterar utställningens estetik bättre.

3.2 RESTful API

Som nämnt under föregående underrubrik sker uppladdningen av bilderna över HTTP-protokollet. Av anledning att göra systemet så generellt som möjligt skapas ett *RESTful API*⁴ för hanteringen av bilder. Gränssnittet behandlar exempelvis att visa, ta bort, ladda upp samt analysera och manipulera bilder. Detta gränssnitt möjliggör för fler tillämpningar än den nu existerande. Det kan användas till egentligen vilket typ av galleri som helst och från vilken plattform som helst. Gränssnittet är programmerat i PHP och eftersom PHP har relativt dåligt stöd för många av HTTP-verben så används Slim⁵ för att underlätta för kommunikationen över HTTP. Dokumentation för denna del av projektet återfinns i bilaga E.



Figur 3.1: Visar övergripligt hur en klient kommunicerar med API:et samt hur API:et enbart är ett tunt lager som kopplar ihop bland annat data från en databas.

⁴http://en.wikipedia.org/wiki/Representational_State_Transfer

⁵<http://slimframework.com>

3.3 Webbapplikationen

Webbapplikationen är uppdelad i två delar, en publik del och en administrationsdel. Mjukvaran skrivs i HTML, CSS, JavaScript samt PHP.

Källkod specifik för webbapplikationen som exempelvis hanterar bilderna skrivs agnostiskt mot olika ramverk och kan implementeras egentligen var som helst. De skrivs efter väl utvalda designmönster för att möjliggöra detta faktum.

3.3.1 Model-View-Controller

[4]Mjukvaran för hemsidan skrivs enligt Model-View-Controller-mönstret (MVC) för att grovt separera på ansvarsområdena i systemet. Representationen av information separeras i *model*-lagret, vy-specifika delar i view-lagret som ska vara fri från logik. Dessa två delar kopplas samman i *controller*-lagret där information, eller data, binds till en vy som visas för användaren. För hemsidan används, liksom till RESTful API:et, ramverket Slim som är ett minimalistiskt ramverk som bland annat hjälper att hantera begäran och svar till hemsidan med ett enkelt gränssnitt. Gruppen väljer att inte använda ett så kallat *full-stack* ramverk, vilket innebär mer omfattande gränssnitt för webbapplikationer, eftersom det helt enkelt inte behövs för webbapplikationens enkla form och att det dessutom ger mer frihet när det gäller vilka andra tredjepartsramverk som kan användas. Slim agerar som ett hölje och binder samman alla delar för applikationen och agerar naturligt som en slags controller med hjälp av inbyggda funktioner. För model-lagret väljs ORM-ramverket *Eloquent*⁶ som ger ett enkelt, abstraherat gränssnitt för databashantering. Den HTML som applikationen till slut ska visa renderas som mallar med ramverket *Twig*⁷ som är väl inlindat i Slim. Dessa tre delar gör att det finns en typ av MVC-arkitektur för applikationen.

3.3.2 Dependency Injection

Eftersom många av klasserna behöver ha tillgång till databasen anges en klass med ansvar som ett beroende för den klassen. Detta sker med hjälp av *Dependency Injection*⁸, vilket bland annat underlättar testning. Som nämnt innan används Eloquent för databashantering i applikationen, men om de applikationsspecifika klasserna skulle göras beroende av ett tredjepartsramverk är klasserna inte längre agnostiska mot andra ramverk. För att lösa detta problem tillämpas Adapter-mönstret, vilket innebär att specifikation av ett gränssnitt för hur databashantering skall se ut i applikationen, och klasserna tillåts bero av detta gränssnitt. Senare injiceras en adapter, specifik för i detta Eloquent-fall, som uppfyller gränssnittets krav. I termer om PHP gäller detta en klass som implementerar ett *interface*. För att hantera alla dessa dependency injections återfinns en fabriksklass som hanterar instansiering av klasser (*Factory*-mönstret). Ett annat vanligt sätt att hantera detta på är att skapa en behållare för alla klasser och deras respektive beroendenden, en så kallad *Inversion of Control Container*.

3.3.3 Offentlig del

På den publika delen av hemsidan finns ett galleri med bilder som användare har målat samt en sida som illustrerar statistik över användandet och de bilder som laddats upp. När en användare sparar en bild vid stationen får denne en webbadress till verket på hemsidan. Via den länken kommer användaren till en sida där bilden kan hämtas. Där kan även konstverket och konstnären namnges, bilden kan

⁶<http://laravel.com/docs/eloquent>

⁷<http://twig.sensiolabs.org>

⁸<http://martinfowler.com/articles/injection.html>

delas på Facebook samt tillåtas att visas publikt på hemsidan. Om användaren tillåter att bilden publiceras kommer bilden automatiskt att visas i galleriet. Första gången en användare använder denna länk, som har ett unikt alfanumeriskt id på fem karakterer, får denne administrera bilden och efter detta kommer användaren behöva ett längre (40 karakterer) id för att kunna ändra informationen för sitt konstverk. Denna presenteras för användaren vid första besöket. Detta för att slippa behöva registrera sig men ändå kunna administrera sina konstverk. Efter första visningen på den korta webbadressen kommer den inte längre visa administrationsdelar utan enbart konstverket.

3.3.4 Administrativ del

För att undvika att olämpliga bilder visas i galleriet har administratörer möjlighet att logga in på hemsidan för att administrera konstverken. När en administratör loggar in visas en meny med fem alternativ: "Obehandlade bilder", "Godkända bilder", "Blockerade bilder", "Taggar" samt "Logga ut".

Under fliken "Obehandlade bilder" finns bilder som användarna väljer att publicera, men som inte blivit granskade av en administratör. Genom att markera de bilder som anses vara olämpliga och opassande för galleriet kan administratören göra dem oåtkomliga för allmänheten.

Bilder som blivit granskade och godkända hamnar under "Godkända bilder" och de som blivit blockerade hamnar under "Blockerade bilder". Ifall det blir aktuellt kan en administratör godkänna en bild som blivit blockerad och vice versa.

Administratörer kan ge bilder olika taggar för att beskriva vad bilden föreställer. Till exempel kan en bild med motivet av en kyckling få taggarna "kyckling", "påsk" och "fågel". Dessa taggar ska kunna användas för att begränsa vilka bilder som visas i galleriet. Ett exempel på detta skulle vara att endast visa bilder med taggen "jul" under december månad.

Webbapplikationens kod enhetstestas med hjälp av PHPUnit som är ett ramverk för just enhetstestning av PHP. Dessa skrivs genom att ställa upp scenarien för användning av exempelvis klassen som ska testas och jämföra resultatet med ett förväntat resultat. En klass har en motsvarande testklass med metoder som beskriver dessa scenarien (test). Alla test körs via kommandotolkten.

3.3.5 Distribuera

Webbapplikationen ligger på ett webbhotell där den har FTP- och SSH-åtkomst. Denna typ av värd är inte direkt optimal eftersom applikationen begränsas av servern. För just distribution av webbapplikationen är det typiska tillvägagångssättet för denna typ av värd en FTP-klient laddas ner och installeras, sedan dras filerna över till servern. I detta fall finns som sagt SSH-åtkomst, vilket gör att vissa trick kan utföras med exempelvis kommandotolkprogrammet **rsync** för att synkronisera filer som kan filtreras efter specifikation med endast ett kommando.

3.3.6 Galleri

I galleriet kan besökare beskåda alla de bilder som målats vid stationen och blivit godkända för publik visning. Bilderna visas i ett bildspel som det antingen går att bläddra manuellt eller automatiskt i. Bläddrar användaren manuellt görs det via knapptryck på pilar. Automatisk visning är standard när sidan laddas. En bild visas i 7 sekunder, sedan bläddrar bildspelet vidare till nästa bild. Den automatiska bildvisningen går att pausa.

Galleriet skrivs i HTML, CSS och JavaScript. Vid uppstart av skapandet av bildspelet hämtas inspiration från noobSlide⁹ som är ett MooTools-baserat bildspel. Detta galleri anpassas sedan efter ändamålet och en ram är bland annat tillagt runt bildspelet, som ska påminna om en tavelram, just för att framhäva att det är en utställning. Bilderna till galleriet hämtas från databasen.

3.3.7 Statistik

Bilden analyseras genom att läsa av alla pixlar i bilden för att avgöra om var pixel är av färgen röd, orange, gul, grön eller blå. Från denna information räknas sedan den procentuella färgfördelningen ut för bilden, vilket tillsammans med övrig information om bilden sparar i databasen. Denna information tillsammans med exempelvis datum ger oändliga möjligheter att representera data där endast fantasin sätter gränserna.

Informationen i databasen används på statistiksidan för att redovisa två olika sorters data. Den första grafen är ett cirkeldiagram som presenterar datan för den totala färgfördelningen hos samtliga bilder som målats och den andra är ett linjediagram som visar färgfördelningens variation för de senaste sju dagarna. Dessa diagram illustreras genom att använda rådatan med biblioteket Chart.js¹⁰ canvas-elementet i HTML5.

3.3.8 Databashantering

I databasen för webbapplikationen finns fyra tabeller. En för bilderna, “image” samt en för taggarna, “tags”. Då en bild kan ha flera taggar och en tagg kan tillhöra flera bilder har dessa tabeller en *Many-to-Many* relation. För att beskriva detta förhållande behövs det en så kallad *pivottabell* som kopplar ihop bilderna och taggarna, “image_tag”. Slutligen finns det en tabell för administratörer, “user”.

<i>images</i>	<i>image_tag</i>	<i>tags</i>
<pre> id : int name : string hash : string given_name : string artist : string status : int published : bool colors : JSON created_at : date updated_at : date viewed_at : date </pre>	<pre> image_id : int tag_id : int </pre>	<pre> id : int name : string created_at : date updated_at : date </pre>

Figur 3.2: Tabellerna i databasen som beskriver ett Many-to-Many-förhållande med pivot-tabellen i mitten som beskriver förhållandet mellan bilder och deras taggar och vice versa.

Ovan visas en bild (Figur 3.2) över tabellernas struktur. Varje bild och tagg som sparas har ett unikt id, dessa kopplas sedan samman i “image_tag”. “image_tag” ser även till att en bild inte kan bli taggad med samma tagg fler än en gång. Som tidigare nämnt används ramverket Eloquent för kommunikation med databasen. För varje tabell i databasen skapas en motsvarande model som interagerar med tabellen. Detta gör att kommunikationen med databasen blir lättformulerad.

För att garantera att samtliga utvecklare har samma data och samma inställningar för databas i deras lokala utvecklingsmiljö skapas en databas *migration*. Detta begrepp kan innebära flera snarlik saker,

⁹<http://www.webappers.com/2008/04/14/noobslide-mootools-slide-show-image-galleries/>

¹⁰<http://chartjs.org>

i detta fall innebär det ett schema för hur databasen ska se ut för webbapplikationer samt funktionalitet att upprätta databasen enligt detta schema. Funktionen att fylla databasen med falsk data för testning lokalt existerar likaså. Allt detta med ett kommando.

3.3.9 Dokumentation

Kommentarer för samtliga PHP-delar skrivs enligt Doxygen-kompatibla dokumentationsblock som, tillsammans med förklarande text, beskriver parametrar och returvärde för metoder och funktioner på ett strukturerat sätt. Detta skapar möjligheten att generera dokumentation för hela systemet utifrån källkoden. All kod för projektet ska vara väl dokumenterad och detta granskas av gruppmedlemmar, vilket definieras av projektets publiceringsrutiner.

Kapitel 4

Resultat

4.1 Vattenfärgssimuleringen

Då applikationen inte längre finns tillgänglig som utställning på grund av hårdvaruproblem kan inte heller användartester utföras på stationen. Testerna utförs istället på en separat dator där tracking av penseln sker med muspekaren. Implementationerna med knappar för att ladda ner och påbörja ny bild fungerar som planerat. Detsamma gäller även utskrift av länk till webbadress för nedladdning av bild.

Genom att först föra penseln mot glaset och sedan valfri färg beräknas den kulör som sedan visas när penseln förs över pappret. Precis som vid målande av riktiga vattenfärgar tar färgen på penseln efter ett tag slut, och nytt vatten och färg måste således hämtas för fortsatt skapande. Genom att dra penseln över flera färgpuckar blandas kulören. Då en konstnär är nöjd med sin bild kan denne välja att antingen spara eller påbörja nytt verk. Detta görs genom att hålla penseln ovan vald knappikon. Om penseln placeras för att spara visas en länk som användaren kan använda för att administrera bilden online. Om penseln ändå placeras vid val av ny bild ersätts den målade bilden med en obehandlad yta. Knappikona reagerar bara efter en viss tid, på så sätt förhindras önskade knapptryck.

Resultatet av den färdiga VR-vyn visas i (figur 4.1). Utseendet av vattenfärgsapplikationen ger ett intryck av en trevlig, verklighetsanknuten miljö att måla i.

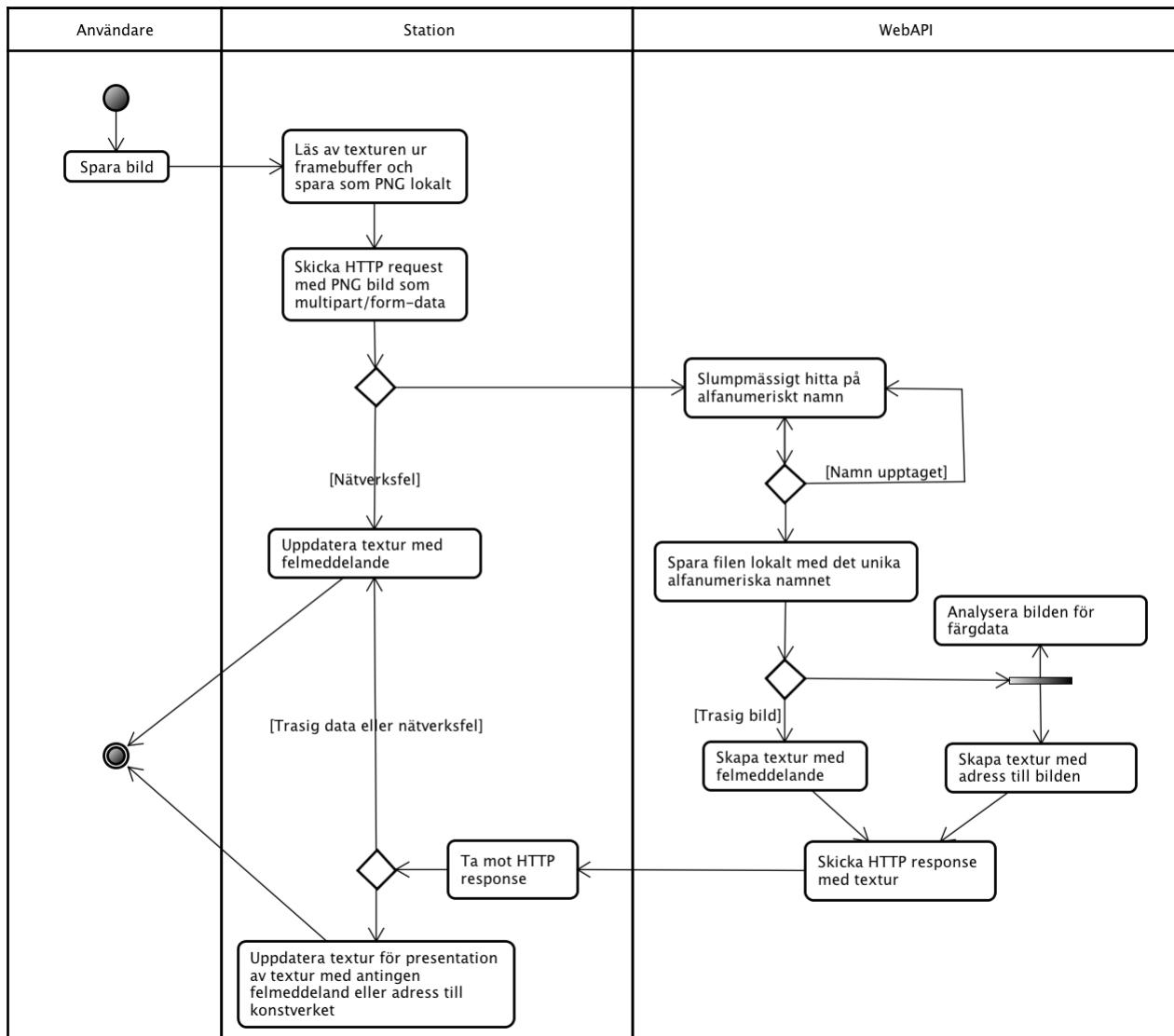


Figur 4.1: Målarstationen

4.2 Integration mellan station och webbapplikation

Kommunikationen mellan målarstationen och webb-applikationen fungerar som önskat.

In nuläget analyseras bilden i samband med att den sparas, vilket leder till att det tar ett par sekunder innan användaren får länken till sin bild. Denna väntetid skulle kunna förkortas genom att låta datan från de bilder som sparats först läggas till i databasen för att sedan låta bilden ställas i en kö för analys.

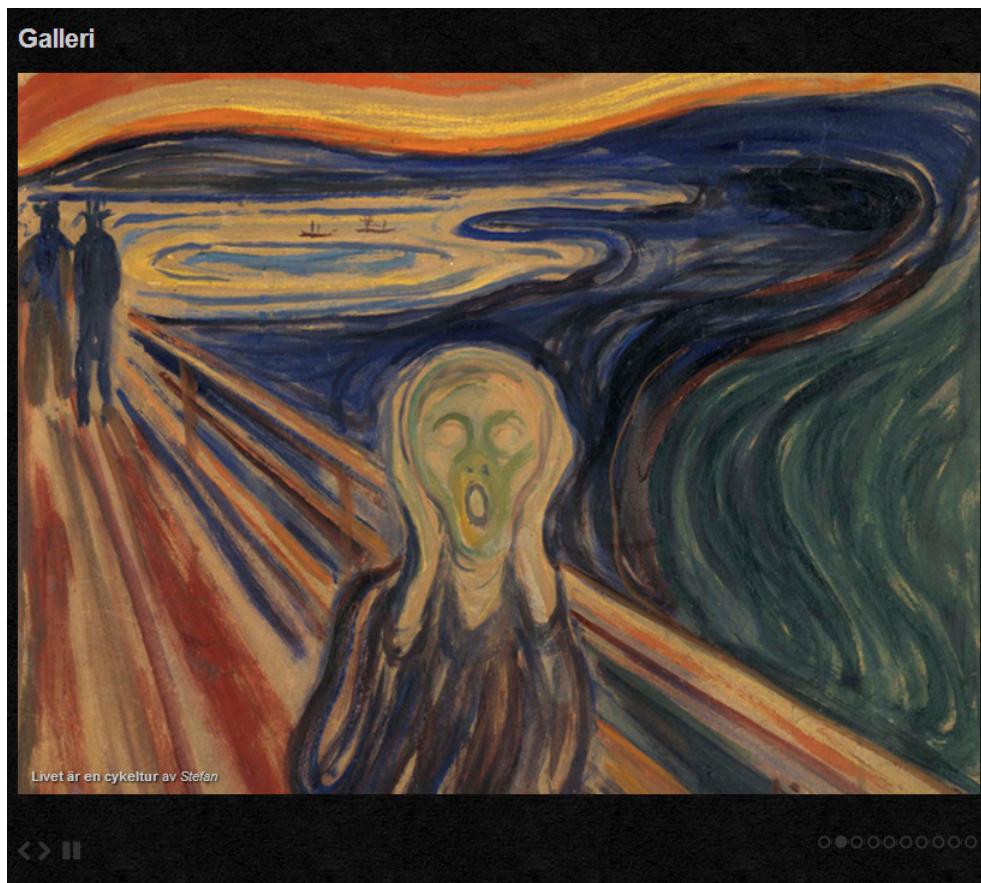


Figur 4.2: Figuren ovan visar kommunikation och separation för stationen och webbapplikationen.

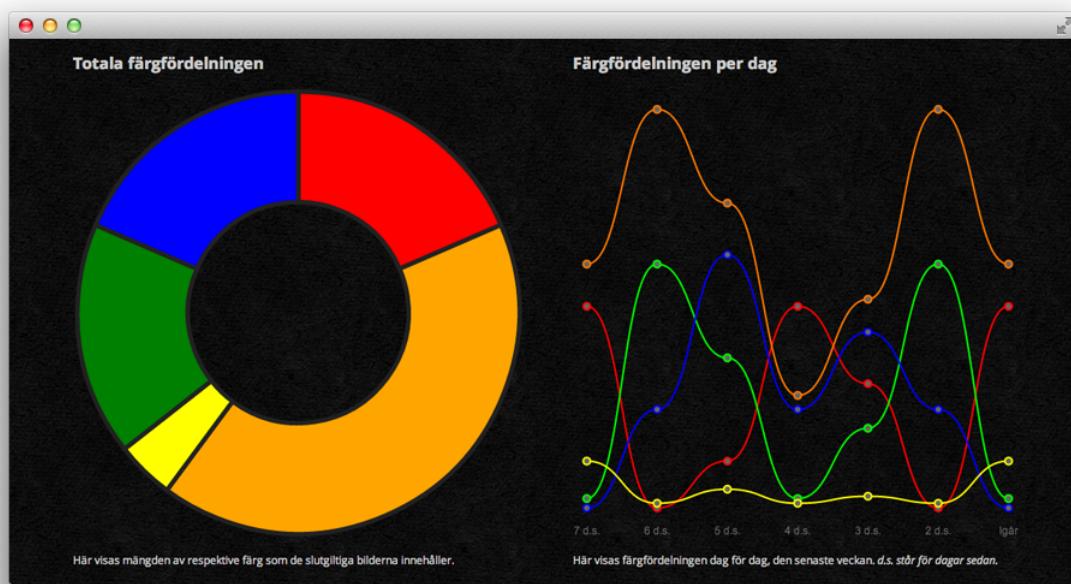
I figur 4.2 visas vilka olika ansvar stationen och webbapplikationen har samt en robust felhantering. Om kommunikationen någon gång misslyckas kommer alltid användaren få veta detta via ett felmeddelande. Fel kan ske på flera ställen och därför finns skydd för detta både i stationens och webbapplikationens mjukvara.

4.3 Webbapplikationen

Hemsidan som utvecklingsgruppen har bearbetat finns att besöka på www.vis-c.se. Där finns både galleri och statistik att beskåda. I galleriet (figur 4.3) syns de bilder som godkänts för publicering av både konstnär och administratör. Statistiken visar den totala färgfördelningen samt färgfördelningen per dag hos de publicerade bilderna, se figur 4.4 nedan.



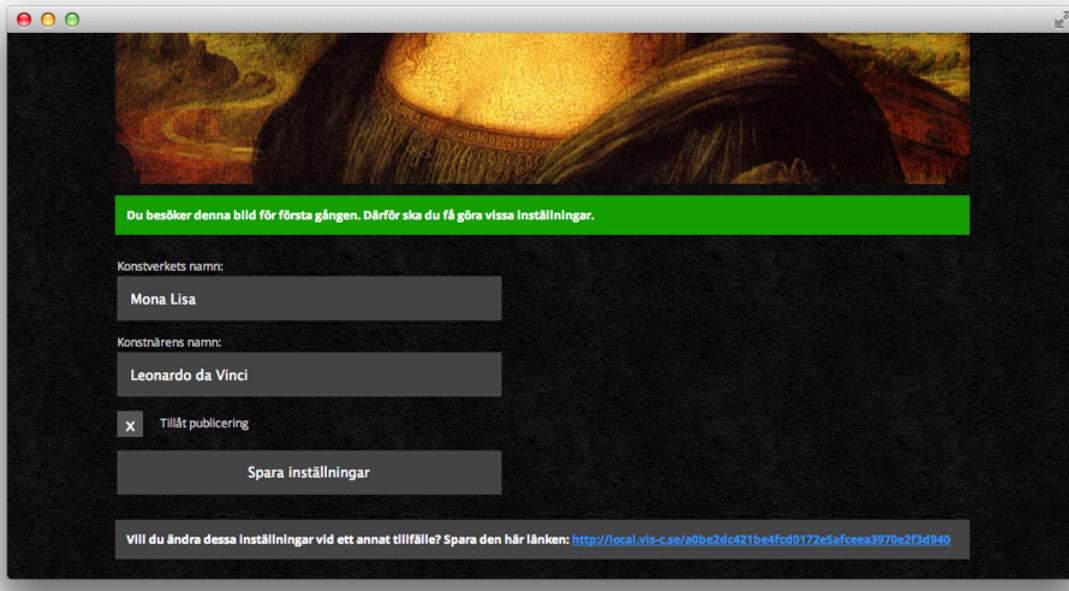
Figur 4.3: Galleri



Figur 4.4: Statistik

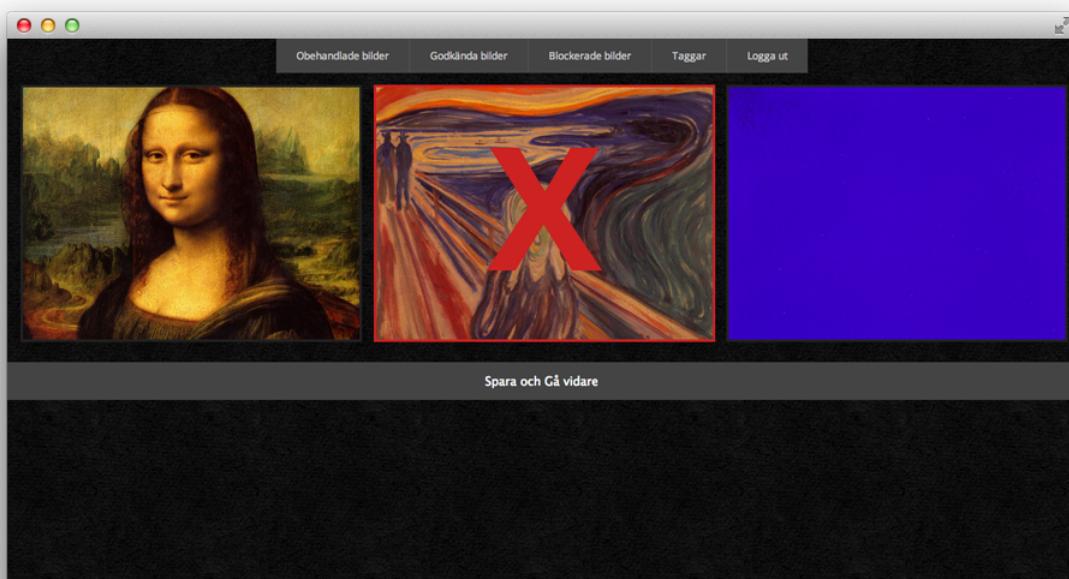
När en besökare väljer att spara sin bild vidarebefordras denne till länk där både konstnären samt konstverket kan namnges. Bildens skapare kan även välja om bilden tillåts publicering eller inte, men

det är alltså möjligt att hämta sin bild oavsett vad, se figur 4.5.



Figur 4.5: Vyn för bilden då användaren besöker den för första gången.

Med hjälp av inloggningsuppgifter kan administratören logga in på hemsidan. Det är här bilder godkänns och blockeras. Alla icke-behandlade bilder hamnar under en egen flik, likaså de godkända samt blockerade bilderna. Administratören kan välja ut bilder att behandla genom att klicka och markera dem, dess vy iakttas i figur 4.6.



Figur 4.6: Administratörs-vy

Förutom att blockera/godkänna bilder kan även administratören addera ”taggar”, det vill säga ord som beskriver bilden. Detta sker under en egen flik kallad ”Taggar”.

Kapitel 5

Diskussion och framtida arbete

Det finns flera faktorer som påverkar resultatet, inte minst att det från början redan fanns en färdig produkt för själva vattenfärgsmålandet som satte begränsningar lika väl som möjligheter för utvecklingen. Stationen på Visualiseringsscentret lades ned medan projektarbetet fortlöpte och därav genomfördes aldrig något användartest på stationen. Därför fick gruppen inte någon inblick gällande hur den faktiskt skulle fungera i praktiken. Möjligheten fanns att testa mjukvaran på en extern dator med muspekaren som tracking. Ett test genomfört på detta sätt skulle dock sakna en viktig dimension, nämligen centrets målarbänk och dess komponenter. Gruppen valde därför att inte genomföra ett sådant test. En inblick från ett användartest hade dock möjligt kunnat påverka nuvarande resultat då besökarnas upplevelse utgör en viktig del av stationen.

Så länge stationen på Visualiseringsscenter C är nerlagd lönar sig inte fortsatt utveckling av projektet. Visserligen kan grundidéerna samt www.vis-c.se tillämpas på andra liknande användningsområden, men det är inget som utvecklarna väljer att aktivt bearbeta.

Kravet att följa en viss typ av arbetsmetod har varit både lärorikt och givande. En utmaning till en början var att begripa alla de termer som *Scrum* innefattar, projektet hade kunnat ta fart tidigare om dessa kunskaper funnits från start. Anledningen till att agil utveckling har kommit att bli ett populärt inom just systemutveckling har blivit allt tydligare under arbetsprocessen då den iterativa processen har visat sig vara väldigt effektiv.

Då en del i gruppen inte hade jobbat med git sedan tidigare tog det tid att komma igång med det. Vanan att alltid se till att ha den senaste uppdateringen av det som finns på exempelvis GitHub är för flera utvecklare ett helt nytt sätt att arbeta. För att undvika att kodgranskaren får alltför många filer att granska på en gång bör pull requests skickas ofta, även detta var något som slarvades med till en början. Trots detta är gruppen överens om att det hade varit svårt att hitta annat komplement som hade fungerat bättre än GitHub.

Gruppen lyckas tillämpa *Scrum* i stora drag, men inte fullständigt. Ofta tar en *task* mer än en arbetsdag att utföra och det nerlagda arbetet registreras inte alltid. Ibland bearbetas ett moment som saknar *task*, vilket gör att Agilefant stundtals ger missvisande information gällande utvecklarnas nerlagda arbetstid. Det är svårt att planera hur lång tid ett arbete tar att utföra när man inte har kunskapen av de verktyg som ska användas eller den kompetens som krävs för att direkt veta hur ett visst problem kan eller ska lösas. För att bli bättre på detta krävs det erfarenhet och det är en sak som man självklart kan få mer av och bli bättre på att planera ju mer man arbetar med exempelvis mjukvaruutveckling.

Tidsplanen följs inte enligt den mall som infattades vid projektets början, främst på grund av att gruppen inte alltid har hunnit avsluta sina *sprint* under utsatt tid. Detta har dock fungerat bra i och

med att sprintsen fortlöper med en veckas mellanrum där var utvecklare får tid att färdigställa det som blivit gjort.

Kapitel 6

Slutsatser

Gruppen har fokuserat på att göra målarstationens användargränssnitt så lättförståeligt som möjligt. Detta för att eliminera risken att användaren exempelvis råkar radera konstverket istället för att spara det. Då slutprodukten inte kunnat testas på dess avsedda målgrupp är det svårt att avgöra huruvida gränssnittet är lättförståeligt eller ej. Detta medför även att gruppen inte kan fastställa ifall målarstationen har blivit mer attraktiv i detta avseende.

En fungerande interaktion mellan målarstationen och publiceringsplattformen har implementerats med hjälp av ett RESTful API.

Då den ursprungliga koden trots allt var fungerande lät utvecklingsteamet bli att ändra den för att istället prioritera utvecklingen av en väl fungerande webbapplikation. Utvecklarna är dock övertygade om att koden kan nyttjas för ytterlig utveckling av stationen, exempelvis vid kundens tidiga önskemål om ökad precision i målandet. Koden är dessutom i behov av refaktorering ty den vid kompilering ger ett flertal varningsmeddelanden.

Något som utvecklarna kunde dragit nytt av under utvecklingsprocessen är skapandet av ett klassdiagram över hur klasserna som används samverkar. Då gruppmedlemmarna var insatta i olika delar av projektet har det stundtals varit komplicerat att begripa den dokumentation som förs. Ett diagram hade då varit ett användbart komplement för detta.

Fastställas kan att kundens krav är uppfyllda och projektet har således fullföljt sitt mål.

Det har varit lärrikt att arbeta med agil utveckling. Inledningsvis var det tidskrävande att tillämpa alla metoder, men det har samtidigt gett mervärde till utbildningen. Om projektet skulle startas om på nytt med den erfarenhet gruppen nu besitter skulle arbetsprocessen fortgå smidigare.

Valet att använda Agilefant till kravhanteringen visade sig inte vara optimalt. Agilefant lämpar sig bättre för större projekt där arbete utförs dagligen. Det fanns många funktioner som gruppen inte kunde använda eller utnyttja fullt ut. Om gruppen hade gjort om projektet igen hade de istället använt Trello som lämpar bättre sig för projekt i denna skala.

Litteraturförteckning

- [1] Lucas Correia De Verdier och David Jonsson, *A Real-time Watercolor simulation for a Novel Tracking Device*, ITN 2012
- [2] Shari Lawrence Pfleeger och Joanne M. Atlee, *Software Engineering, Fourth Edition, International Edition*, Pearson 2010
- [3] Ken Schwaber och Jeff Sutherland, *Scrumguiden, Den definitiva guiden till Scrum: Spelets regler*, Juni 2013, [www] <https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide-SE.pdf#zoom=100>
- [4] Ian Davis, *What Are The Benefits of MVC?*, december 2008, [www] <http://blog.iandavis.com/2008/12/09/what-are-the-benefits-of-mvc/>

Bilaga A

Individuell prestation

A.1 Tim Brodin

Har varit produktägare och därmed ansvarat för att backlogen har varit uppdaterad och korrekt. Vidare har Tim stått för kommunikationen med kund. Som medlem i utvecklingsteamet har han arbetat med vidareutveckling av målarstationen. Där har användnings- och interaktionsmöjligheter undersökts och en grund till ett GUI skapats. Tim har även bidragit med statistik till webbapplikationen där han har skrivit och implementerat både uträkningar och visualiseringar av data.

A.2 Alexander Cederblad

Har haft ansvaret för kodgranskning i utvecklingsgruppen och ser till att all kod är fungerande och välskriven. Har skrivit implementationen för att kommunicera med REST API i utställningsapplikationen, inklusive detta API som behandlar bilderna. Har ställt upp utveckling och testningsmiljön för webbapplikationen i PHP. Har varit involverad i samlig kod för webbapplikationen framför allt back-end, men även en betydande del front-end.

A.3 Kristina Engström

Har haft rollen som koordinator i gruppen och därmed sett till så att gruppen har någonstans att arbeta och mötas. Har i tidigt skede arbetat med GUI för den publika delen av webbapplikationen med Sofia. Kristina implementerat taggnings-funktionen samt upprättat en mall för projektrapporten i L^AT_EX.

A.4 Sofia Wennström

Har upprättat GUI för samtliga vyer på hemsidan (förutom statistik) samt knappanelen i målarapplikationen. Har även hittat lösningar för delning med sociala medier och markering av bilder. Vid mötena har Sofia fört mötesanteckningar, hon är dessutom rapportansvarig i teamet. Som rapportansvarig ser hon till att alla krav uppfylls och att språket hålls formellt. Hon har framför allt arbetat med HTML- och CSS-, men även en liten del javascript- samt PHP-programmering.

A.5 Anton Österblad

Anton har haft rollen som projektledare och har ansvarat för att gruppen följer de angivna metoderna samt håller sig till tidsschemat. I utvecklingsteamet arbetade Anton inledningvis med funktionen

att spara bild i utställningsapplikationen samt med utredning av lagringsfunktioner och tekniska lösningar för anslutning till internet från stationen. Efter det arbetade han främst med galleriet i webbapplikationen.

Bilaga B

Projektplan

B.1 Kort beskrivning

Virtuella vattenfärgar är en realtidssimulering av vattenfärgar som fanns som utställning på Visualiseringscenter C. För att måla används en pensel (trackingenhet) som doppas i ett glas med virtuellt vatten och sedan doppas i ett urval färger. Man målar mot en yta där resultatet projiceras i realtid. Färgen beter sig verklighetstroget beroende på hur mycket vatten och färg man tar. Om penseln ej doppas i vattenglaset går det ej att måla.

Kund är Anna Öst som jobbar på Visualiseringscenter C. Uppdraget är att vidareutveckla ursprungsprodukten. Något som idag saknas är möjligheten att spara och publicera målningar.

Målgruppen för utställningen är förskole- och lågstadieelever, vilket är väldigt viktigt att tänka på under hela utvecklingsprocessen.

Mål för projektarbetet:

- Att lära sig jobba med systemutveckling i en projektgrupp.
- Vi ska göra utställningen Digitala Vattenfärgar mer attraktiv.

B.2 Teknisk beskrivning

Projektet hör till en utställning på Visualiseringscenter C som utvecklades som ett examensarbete med rapport *A Real-time Watercolor Simulation for a Novel Tracking Device*, 2012 av Lucas Correia de Verdier och David Jonson vid Linköpings Universitet. I rapporten beskrivs i viss grad det system som används i utställningen. Detta projekt är en vidareutveckling och påbyggnad till den utställningen.

Systemet för utställningen kommer att behöva vara väldigt stabilt och kunna köras under hela dagar, sju dagar i veckan. Det behöver dessutom vara enkelt att administrera systemet för gemene man.

Trackingsystemet som används bygger på att mäta magnetfält och kräver därför frånvaro av magnetiska material i största möjliga utsträckning. Trackingsystemet är utvecklat av en forskningsgrupp vid Linköpings Universitet.

Mjukvara för simuleringen är utvecklad i C++, med Boost, OpenGL och GLSL, i *Visual Studio 2010* och körs därmed på Windows-hårdvara. Själva fluidsimuleringen för vattenfärgerna körs på grafikkortet.

Systemet för att kunna spara och ladda upp bilder på Internet för inspektion och senare utställning kommer kräva en kommunikation mot Internet från mjukvara. Mjukvara för granskning av bilder kommer att vara av typ webbapplikation för portabilitet och användarvänlighet. Bilder kommer behöva lagras någonstans på Visualiseringscenter C alternativt i molnet.

Beroende på hur alla bilderna kommer lagras kan en lösning såsom *cURL* användas för att läsa/skriva över FTP.

I nästa del av projektplanen beskrivs olika lösningar för hur användaren ska få tillgång till bilden efter att ha gjort valet att spara en bild. För QR-kod kommer ett tredjeparts API behöva användas som kan koda om en webbaddress till just en QR-kod.

B.3 Infrastruktur och systemutveckling

B.3.1 Utställningsmonter

Mjukvara för själva utställningen kommer utvecklas för Windows, då det redan existerande systemet kommer vidareutvecklas.

B.3.2 Bildpublicering och inspektion

Mjukvara för bildinspektion och utställning (bildspel) av konstverk skapade vid utställningen kommer vara en webbapplikation och kommer kunna köras i alla moderna webbläsare.

Utställningens mjukvara måste tillåta att skicka bilder via Internet och ett användargränssnitt för att publicera bilder. Nedan finns ett antal olika lösningar, med respektive för- och nackdelar, för publicering av bild:

- QR-kod
 - + Enkelt att gå till hemsida.
 - + Relativt enkelt att implementera.
 - Få användare som har QR-läsare till hands. Jobbigt att ladda ner QR-läsare.
- Visa webblänk till bilden (ex. visc.se/5A7B1)
 - + Man måste gå till hemsidan manuellt för att se sin bild.
 - + Bra som första implementation.
 - + Väldigt enkelt att implementera.
 - Kräver manuellt arbete på ex. telefon.
- Tangentbord för att skriva in e-mailadress
 - + Man får bilden direkt till sin inkorg.
 - + Man slipper göra manuellt arbete på mobiltelefon.
 - Relativt svårt att implementera.
- Utskrift på plats
 - + Man får bilden på papper.

- + Kan vara vinstmaskin om man tar dyrt betalt.
- Kräver administration.
- Toner/färgpatroner/skrivare kostar mycket pengar.

- *Bluetooth*

- + Kräver inget manuellt skrivarbete på telefonen.
- Gemene man använder inte Bluetooth ofta.
- Relativt svårt att implementera.

- Ett urval av ovanstående alternativ

- + Fler alternativ
- Fler alternativ

B.3.3 Stories och tasks

Varje *story* kommer skapas utifrån kund eller vår egen tanke om projektets resultat. Dessa *stories* kommer att delas upp i flera *tasks* som kan utvecklas och färdigställas under en *sprint*. Från kund har vi fått dessa önskemål:

- Spara och publicera sina målningar.
- Att kunna visa upp utvalda målningar på en skärm på C.
- Att kunna filtrera bort svordomar och obscena bilder.
- Ökad precision i målandet:
 - Möjligheten att välja mellan olika penslar.
 - Mer detaljer, men behålla realismen.
 - Annan effekt, ex. blyerts.
- Kunna visa målarprocessen i form av en video.
- Föra och visa statistik över hur utställningen används.

Vi anser att huvuddelen i projektet består av att implementera ett väl fungerande publiceringsverktyg där man kan hämta hem de bilder man målat. Det ska finnas möjlighet att publicera konstverk för utställning på Visualiseringsscentret samt tillhörande administrationsverktyg för att rensa ut olämpligt innehåll. Möjligheten att gilla dina egna och andras bilder ska även vara tillgänglig.

Vi vill även beräkna olika sorters statistik exempelvis vilken färg som används mest, hur många bilder som sparas per dag och hur lång tid spenderas vid stationen. Statistiken ska sedan vara tillgänglig för administratören.

B.3.4 Versionshantering och kodkvalité

Projektets agila del av utvecklingen kommer att administreras med hjälp av webbapplikationen *Agilefant*. Källkoden för mjukvara kommer versionshanteras med git, med ett *repository* på *GitHub*. Detta för att *GitHub* är i våra ögon de facto standard och möjligheten att använda deras *Issue tracker* för att kunna jobba tillsammans utan problem ur ett kodgranskningsperspektiv.

Eftersom vi redan har fått källkod från det ursprungliga projektet så kommer vi lägga så lite fokus som möjligt på att refaktorera den koden och fokusera på vår egna kod. Vi kommer lägga vikt i att skriva test för vår egen mjukvara i form av enhetstest. Troligtvis kommer en hel del regressionstest behöva skrivas då våra implementationer till programmet kan skapa problem i den gamla koden och i sin tur hela programmet. Gruppmedlemmarna kommer att manuellt granska varandras kod för att garantera läslighet och kvalité. Körttest kommer att genomföras efter slutförd *sprint*. Test av färdig programvara kommer kunna göras publikt på stationen på Visualiseringsscenter.

Övrig dokumentation delas gemensamt på *Google Drive*. Dessa dokument kan vara tankar och idéer som vi diskuterar tillsammans. De dokument som uppdateras kontinuerligt kommer sparas med datumstämpel för varje version. Om inte *Google Drive* är tillräckligt i längden kan det diskuteras att spara dokument på *GitHub* likaså, möjligtvis som *Markdown*-dokument.

Mer formella dokument såsom projektrapporten som är en del av examinationen kommer skrivas i *LATEX* och kommer att versionshanteras på *GitHub*.

B.3.5 Kodgranskning

Nedan följer en lista med saker som ska uppfyllas för att ett kodstycke ska godkännas som körbart kod i produktionsmiljö.

1. Koden följer specificerad standard (*style guide*).
2. Koden har enhetstester för åtminstone den viktigaste funktionaliteten.
3. Koden har kommentarer som beskriver funktionalitet samt författarens tankar i den mån att det är enkelt att förstå för andra.

B.3.6 Projekthantering

PHP

Composer kommer användas för PHP i den utsträckning det går. Detta gör att koden blir lite enklare att hantera.

B.3.7 Kodstandarder

När vi skriver kod ska vissa standarder upprätthållas för att det, bland annat, ska vara enkelt att läsa och förstå koden. Dessa är olika för varje språk.

C++

Google C++ Style Guide

PHP

FIG PSR-2

HTML/CSS

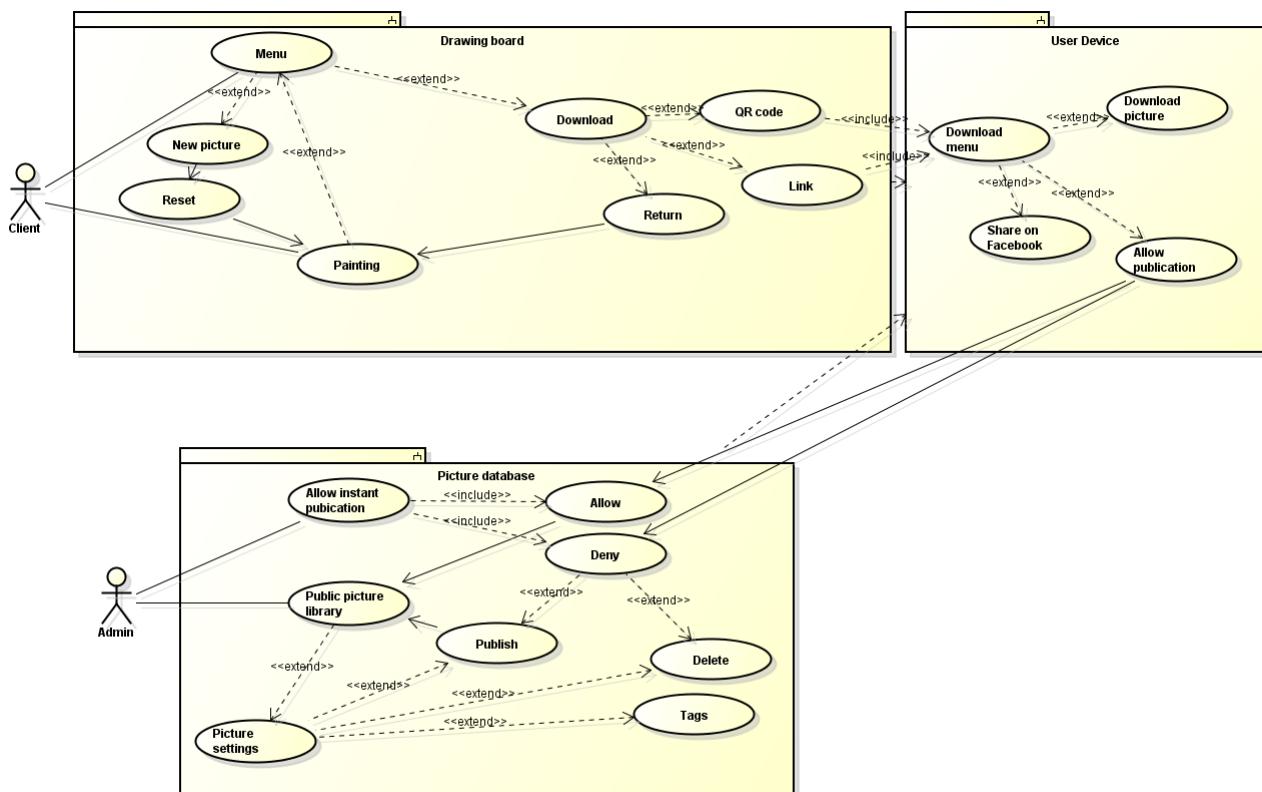
Google HTML/CSS Style Guide

B.3.8 Kravhantering

Kravhanteringen kommer att behandlas i en *backlog*. I backlogen kommer alla önskemål och tillägg till utställningen finnas i punktform tillsammans med deras respektive krav. Det är viktigt att varje punkt har väldefinierade krav. Denna *backlog* kommer utvecklas och skrivas om under projektets gång. I backlogen kommer alla punkter rangordnas efter hur viktig ett tillägg eller en ändring är. Inför varje *sprint* kommer punkter ur backlogen väljas ut för att implementeras under kommande *sprint*.

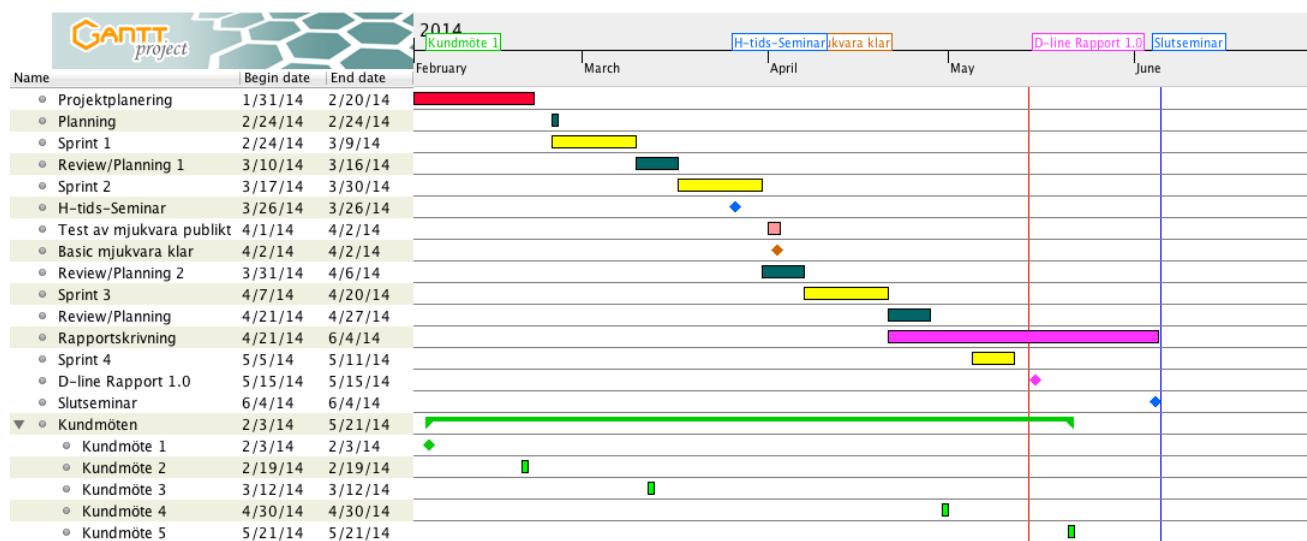
B.3.9 Modelleringsstandard

En modell av systemet har konstruerats med hjälp av UML- och modelleringsverktyget *Astah*. Systemet består av tre avdelningar; Ritbord, användarenhet och bilddatabas. Både användare och administratör ska på olika sätt kunna interagera med systemet.



Figur B.1: Use-Case Diagram

B.3.10 Tidsplan



Figur B.2: Tidsplanen

Bilaga C

Sprints

- Story
 - Task
 - * Beskrivning

Färgkoder:

Klar

Påbörjad

Uppskjuten

Eliminerad/borttagen

C.1 Sprint 1: Stationen

- Spara-funktion
 - Spara bilden, screenshot
 - * Möjligheten att spara en bild i utsällningsapplikationen.
 - Undersöka och skapa lagringsmöjligheter.
 - * Skapa en databas, filsystem.
- Enkelt användargränsnitt
 - Få upp en knapp i scenen.
 - * Fungerande knapp till trackingsystemet med actionlistener.
 - Ruta för QR-kod och länk.
 - * Då användaren vill spara sin målning ska en bild med en QR-kod och länk visas så att användaren kan hämta sin bild.
 - Grafik till knapparna
 - Skapa ett gränsnitt i koden för användargränsnittet
 - Undersöka hur trackingsystemet kan användas

C.2 Sprint 2: Webbapplikationen

- Föra enkel statistik
 - Undersöka möjligheten att föra statistik i koden.
 - * Spara statistik från målandet på stationen.
 - Undersöka analys av sparad bild.
 - * Spara statistik från den färdiga bilden
 - Mäta tiden man målar
 - Föra statistik över vilka färger som finns med i bilden.
- Spara bild cURL/PHP API
 - Ladda upp bild med FTP/HTTP med cURL
 - Skapa ett unikt id för bilden
 - Hantering av eventuella fel som kan uppstå
 - * Onödigt att ladda upp en bild om något blir fel.
 - Spara meta-information i SQL databas
 - Lägg till och visa textur av bild med cURL sömlöst i applikationen
- Användargränsnitt till webbapplikationen
 - Design av användargränsnitt.
 - * Inkluderar även sidan för nedladdning av bild och val av publicering.
 - Dela till sociala medier
 - * Möjligheten att dela sin bild på t ex Facebook för att visa sina bekanta.
 - Hämta meta/bilder från databas
- Bildspel och gillafunktion
 - Bildspel av alla publicerade bilder
 - * De bilder som användare har gjort och som tillåtits att visas publikt ska visas i detta bildspel
 - Möjligheten att gilla bilder som finns på hemsidan
 - * Ett enkelt knapptryck för att gilla-bilden. Någon slags begränsning så att samma användare inte ska kunna gilla samma bild om och om igen.

C.3 Sprint 3: Administration

- Förbättra utställningsapplikationens gränsnitt
 - * Förfinna utseendet så att den blir mer tilltalande.
- Administration av publika bilder
 - Möjligheten att tagga bilder, databas

- * Kategorisera bilder med taggar. I denna task ligger fokus på hur detta ska implementeras i databasen samt möjligheten att lägga in taggar manuellt.
- Möjligheten att tagga bilder, användargränsnitt
 - * Att kunna lägga till och tagga bilder på administrörsidan.
- Bildvy, användargränsnitt
 - * Bilderna ska visas i ett rutnät av 3 x 3 bilder.
- Bläddra i bildvyn, användargränsnitt
 - * Finns det fler än nio bilder ska nya bilder visas då man scrollar ner eller genom att trycka på en ' nästa' knapp.
- Flikar med osedda, sedda samt blockerade bilder, användargränsnitt
 - * För att orientera admin-sidan finns tre flikar. Första fliken, som även är default-fliken, visar bilder som adminstartören ej kontrollerar, andra fliken visar kontrollerade bilder och den tredje fliken visar blockerade bilder som anses olämpliga att publicera.
- Kontroll bildstatus, databas
 - * I fall användaren tillåter publicering av bild ska den automatiskt komma med i bilden spelet på hemsidan samt visas i fliken för osedda bilder på adminsidan.
- Filtrering
 - * Administratören ska ha möjlighet att blockera bilder som är opassande. Det ska endast krävas ett knapptryck för att bestämma om en bild ska visas publiskt eller ej. Alla bilder har som standard att visas publiskt. De bilder som markeras i bildvyn hamnar bland de blockerade bilderna. De som inte markeras märks automatiskt som sedda. Man ska kunna bedömma i en hastighet av max 0.5 sekunder.
- Logga ut, knapp
 - * En knapp för att logga ut från admin-sidan. Ska endast kräva ett tryck.
- Inlogg
 - Användargränsnitt för inlogg
 - * Med användarnamn och lösenord.
 - Skapa inlogg manuellt
 - * En användare skapas manuellt för att kunna logga in på administrörsidan.
 - Koppla inlogg till databasen
 - Säkert inlogg
- Testning
 - Travis-CI
- Strukturera rapporten
 - Gör en mall i L^AT_EX
 - Gör en mall i Google Drive
- Statistik
 - Gränsnitt

- * Utforma vilken statistik som ska visas på webbapplikationen.
- Hämta statistik från databasen
- Slå ihop kod från tidigare *sprints*
- Refaktorera kod och rensa bland varningar, stationen

C.4 Sprint 4: Sista sprinten

- Rapportskrivning
 - Skriva klart projektrapporten i Google Drive-dokumentet
 - Föraöver all text till L^AT_EX-dokumentet
- Lägg till alla funktioner i användargränstittet
 - Lägga till/ta bort taggar på adminsidan
 - Fungerande filtrering av bilder
 - Bestämma om bilden ska publiceras eller ej
 - Lägga till namn på konstnär och konstverk
 - Ett fungerande galleri
 - Fungerande login
 - Fungerande statistik
 - Ladda ner-knapp
 - * Onödig, då det bara är att klicka på bilden.
 - Dela-funktion
 - Från admin-länk till dela-min-bild-länk
- Bonusfunktioner: I mån av tid
 - Anpassa användargränssnittet efter olika skärmstorlekar

Bilaga D

Backlog

Färgkoder:

Implementerad

Borttagen

- Som användare vill jag kunna ladda hem bilderna jag skapat vid utställningen.
 - Gränssnittet ska vara uppenbart för alla åldersgrupper hur man sparar bilder.
 - Det ska ta max 20 sekunder från att man sparar till att man har bilder i ”handen”.
- Som användare vill jag kunna bestämma om min bild ska få publiceras på visualiseringsscentret eller ej.
 - Ska vara tillgängligt överallt, läs Internet.
 - Ska vara så enkelt som en knapptryckning.
- Som användare vill jag kunna kolla på alla bilder som målats på Internet.
 - Ska finnas i form som bildspel.
- Som utställare vill jag kunna visa upp de konstverk som skapats vid utställningen.
 - Bildspelsfunktion på hemsidan.
 - Man ska kunna välja hastighet på bildspelet.
 - Man ska kunna bestämma ett urval att visa ett visst bildspel ex. på Alla hjärtans dag visar man ett kärleksbildspel.
 - Man ska kunna tagga bilder för att senare kunna skapa bildspel snabbt och enkelt ex. ser man en kyckling så taggar man den ”kyckling”, senare när man ska göra ett påskbildspel så söker man bl.a. på kyckling och får upp bilder på kycklingar.
 - Man ska kunna ta bort en tagg från en bild.
- Som användare vill jag kunna gilla andras bilder (för ex. bilder).
 - Ska inte kräva någon inloggning, typ Facebookautentisering.
- Som användare vill jag kunna dela mina bilder jag skapat på sociala medier.
 - Ska endast kräva en knapptryckning.
- Som användare vill jag kunna navigera fram en flervalsmeny.

- Den ska inte störa målandet, alltså menyn ska inte visas av misstag under tiden man målar.
- Man ska inte “råka komma åt en knapp”.
- Menyknapparna ska vara tillräckligt stora för trackingsystemets upplösning. Hör ihop med punkten ovan.
- Symboler, metaforer, och så lite text som möjligt så att alla, som kan eller inte kan läsa eller inte kan se så bra, förstår vad som händer om man klickar på en viss knapp.
- Som administratör vill jag kunna filtrera bort stötande innehåll.
 - Man ska kunna bedöma i en hastighet av 20 bilder per minut.
- Som användare vill jag efter jag har målat kunna se hela processen.
 - Man ska kunna se minst en bild var tionde sekund (0.1 fps).
- Föra statistik över hur utställningen används.
 - Kunna se hur många penseldrag per dag.
 - Kunna se vilken färg som används mest per dag.
 - Kunna se hur många bilder som publiceras per dag.
 - Kunna se vilken färg som är populärast, baserat på alla bilder i databasen.
 - Kunna se vilken färg som varit populärast per dag den senaste veckan.
 - Hur lång tid det tar att måla en bild.
- Som användare vill jag kunna välja olika storlekar på penslar.
 - Max två knapptryck för att välja penselstorlek.

Bilaga E

RESTful API dokumentation

The API is located at `/api/<version>/` and the current development version is `v1`, use with caution. The following are all the thinks you can do with the API. Be sure to have read the description for each route before you using them.

Upload image

POST `/image`

Parameters:

- `image` [png image] A PNG image.

Post an image for storing and respond with response-image. Pass parameter as `multipart/form-data`.

Get all images as JSON

GET `/json/image`

Parameters: *None*

Show all published images in JSON format.

Show image

GET `/image/:id`

Parameters:

- `id` [string] ID of an existing image.

Show an image of the specified image id.

Show image as JSON

GET `/json/image/:id`

Parameters:

- `id` [string] ID of an existing image.

Show a published image as JSON.

Show dummy response image

GET /dummy

Parameters: *None*

Responds with a PNG image.

Show specific dummy response image

GET /dummy/:text

Parameters:

- *text* [string] The text to show.

Responds with a PNG image.

Show error response image

GET /error

Parameters: *None*

Respond with an error image.