Interaktiv visualisering av HPD-temperaturdata i 3D och 2D

Utveckling av ett webbaserat övervakningsverktyg för långsiktig termisk analys av högpresterande datorkluster

LUNDS UNIVERSITET
Campus Helsingborg

Lunds Tekniska Högskola vid Campus Helsingborg
Institutionen för datavetenskap

Examensarbete:

Olof Svensson

Nadjma Zaher

LTH vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2025

Sammanfattning

Detta examensarbete syftar till att undersöka och utveckla en prototyp för ett visuellt övervakningsverktyg som kombinerar 3D- och 2D-visualiserng för att utöka övervakningen av LUNARC:s serverhall. Målet är att möjliggöra en snabb identifiering av termiska hotspots, resursförbrukning och avvikelser, samt analysera historiska trender.

Utvecklingen av prototypen skedde i en JavaScript-miljö med ramverk som React, Three.js och Chart.js för frontend och Node.js med Express.js för backend. Backend hanterar inläsning av CSV-data och omvandling till JSON, vilket används för att presentera data i det interaktiva gränssnittet. API:er skapades för att effektivt överföra data mellan frontend och backend. Funktionaliteten utvärderades genom demonstrationer och feedbacksessioner i samarbete med LUNARC, ett kompetenscentrum för avancerade beräkningar vid Lunds universitet. Arbetet grundades på en kravspecifikation framtagen genom diskussioner och observationer tillsammans med LUNARC.

Examensarbetet resulterade i en prototyp för ett webbaserat övervakningsverktyg som visualiserar serverhallens rack och noder i en 3D-miljö, med färgkodade temperaturgradienter och varningssystem. Ett 2D-gränssnitt kompletterar detta med grafer över temperatur, energiförbrukning och arbetsbelastning, vilka tidigare hanterades manuellt via Excel. Det finns även möjlighet att navigera genom historiska data och se ögonblicksdata för vissa tidpunkter. Prototypen är anpassad för att presenteras på storbildsskärm via en webbläsare.

Slutsatsen är att prototypen har potential att kunna användas för att förbättra LUNARC:s övervakningsförmåga och minska risken för driftproblem genom att visualisera data och identifiera avvikelser på ett mer intuitivt sätt. Prototypen bör främst ses som ett underlag för vidareutveckling med fokus på skalbarhet, databaslösningar och ytterligare funktionalitet.

Nyckelord: 3D-visualisering, serverhallar, temperaturövervakning, JavaScript, Three.js

Abstract

This thesis aims to examine and develop a prototype for a visual monitoring tool that integrates 3D- and 2D-visualization to enhance the monitoring of LUNARC's data center. The goal is to enable the rapid identification of thermal hotspots, resource consumption, and anomalies in real-time, as well as to analyze historical trends.

The development of the prototype was carried out in a JavaScript environment, utilizing frameworks such as React, Three.js, and Chart.js for the frontend, and Node.js with Express.js for the backend. The backend processes CSV data and converts it into JSON format for visualization in the interactive interface. API:s were created to efficiently transfer data between the backend and frontend. The functionality was evaluated through demonstrations and feedback sessions in collaboration with LUNARC, a center of expertise in advanced computing at Lund University. The project was guided by a requirements specification developed through discussions and observations made with employees at LUNARC.

The result of the thesis is a prototype for a web-based monitoring tool that visualizes racks and nodes of the data center in a 3D-environment with color-coded temperature gradients and a warning system. A complementary 2D-interface presents graphs displaying temperature, energy consumption, and workload data, which were previously managed manually using Excel. Additionally, users can navigate historical data and view snapshots of the system and its parameters. The prototype is optimized for display on a large screen through a web browser.

The conclusion is that the prototype has potential to improve LUNARC's monitoring capabilities and reduce the risk of operational issues by intuitively visualizing data and identifying anomalies. The prototype serves as a foundation for further development, focusing on scalability, database solutions, and additional functionalities.

Keywords: 3D-visualization, data center, temperature monitoring, JavaScript, Three.js

Förord

Att arbeta mot ett gemensamt mål kan vara både intensivt och utmanande, men nyfikenhet och glädje över att få lösa komplexa problem är det som driver resan framåt. Just detta har vi fått uppleva under detta examensarbete. Vi har även haft förmånen att kombinera teknisk utveckling med kreativ problemlösning. Vi är särskilt glada över hur våra olika perspektiv och kunskaper har vävts samman till en lösning.

I första hand vill vi tacka varandra för ett givande och engagerat samarbete samt för de lärdomar och idéer som har alstrat ur våra diskussioner. Vår vilja att stötta och lära av varandra har varit en av grundpelarna i detta examensarbete samt att lyckas ta oss an varje ny utmaning med beslutsamhet och god planering.

Vi vill även rikta ett stort tack till LUNARC för möjligheten och förtroendet att få arbeta med ett koncept som gav oss friheten att kombinera praktiska och kreativa lösningar inom visualisering. Ett varmt tack till Anders Follin och Jonas Lindemann för värdefull feedback, insiktsfulla diskussioner, tekniska vägledning och tålamod under hela processen.

Slutligen vill vi rikta vårt varmaste tack till våra handledare och examinator, Christian Nyberg och Christin Swahn för värdefull akademisk vägledning. Deras stöd och råd har gett oss riktning och inspiration under arbetets gång.

Olof & Nadjma

Innehållsförteckning

1. Inledning

I takt med att moderna datacenter blir alltmer komplexa ökar behovet av effektiva verktyg för övervakning och analys av systemparametrar. LUNARC, ett kompetenscentrum för högpresterande beräkningar vid Lunds universitet, hanterar en stor mängd beräkningsresurser som genererar omfattande loggdata. Att analysera och tolka denna data för att optimera kylning, energiförbrukning och systemprestanda är en utmaning, särskilt när parametrarna varierar över tid och är spridda över en stor mängd servrar. I följande kapitel presenteras examensarbetes bakgrund och de övergripande mål och problemformulering vilka ligger till grund för utvecklingen av prototypen.

1.1. Bakgrund

LUNARC tillhandahåller beräknings, lagrings och visualiseringsresurser för Lunds universitet. Dessa resurser är placerade i en extern datorhall och övervakas med hjälp av olika typer av loggningssystem. Då resurserna drar mycket ström producerar de också mycket energi som måste tas om hand. Totala antalet servrar är cirka 250 och olika delsystem har olika karakteristik både momentant och över tid. För att få en förståelse för hur el, kyla och servrar fungerar övervakas många parametrar kring detta till exempel:

- In/ut lufttemperaturer på servrar
- Temperaturer på servrarnas CPU:er
- Strömförbrukning
- Last på serverns operativsystem
- Antalet beräkningsjobb på servern

Det kan vara svårt att överblicka alla dessa parametrar på en hög nivå i felsökningssyfte. Det är speciellt svårt att förstå hur dessa parametrar relaterar till placering av utrustning i datorhallen. Även om de rack som servrarna är placerade i är ett slutet & vatten/luft-kylt system kan strålningsvärme och läckage orsaka termiska hot-spots i datorhallen. En möjlig är att studera är en visualisering av de ovanstående parametrarna (eller fler relevanta) i en interaktiv 3D-modell där parametrarna kan visualiseras direkt på utrustningen placerad i hallen. På detta sätt skulle det vara möjligt att enklare identifiera problem som relaterar till servrarnas aktuella

belastningsprofil samt fysisk placering i hallen. Intressant är också att kunna gå bakåt i tiden och identifiera intressanta händelser rent spatialt. Idealiskt vore en lösning, idealiskt med ett enkelt separat gränssnitt, för en större skärm som även är åtkomligt via en webbläsare för mer detaljerade analyser.

1.2. Syfte

Syftet med examensarbetet är att utveckla en prototyp av ett övervakningsverktyg där loggade parametrar kan visualiseras i en interaktiv tredimensionell modell med möjligheter att även undersöka skeenden historiskt: antingen genom integrering med nuvarande undansparade loggdata eller data som sparas från och med implementeringen av prototypen.

1.3. Målformulering

Examensarbetet har för avsikt att uppfylla följande mål:

1. Att undersöka vilka ramverk som finns för att implementera ett tredimensionellt övervakningssystem för övervakade parametrar.
2. Att genom intervjuer och diskussioner ta fram en kravspecifikation för verktyget.
3. Att implementera en prototyp av detta system som kan användas för att övervaka beräkningsresurserna på LUNARC.

1.4. Problemformulering

Följande problem kommer att undersökas:

1. Hur sker övervakningen av resurserna idag?
2. Vilka parametrar skall visualiseras?
3. Vilka verktyg finns det för att inhämta övervakade parametrar?
4. Vilket programspråk skall användas för att implementera systemet?
5. Vilka bibliotek och verktyg finns det för att implementera 3D-visualiseringen?

1.5. Motivering av examensarbetet

Vi har under vårt sökande efter examensarbete dragit oss till projekt där vi upplever att det finns spännande tekniska utmaningar såväl som kopplingar till samhällsnyttiga tjänster. Att få ta del av ett projekt där vi har möjlighet att arbeta i närhet av kraftfulla beräkningsresurser ämnade åt forskning på Lunds universitet, anser vi uppfylla dessa krav.

Detta examensarbete omfattar flera områden av datavetenskap. Det ger oss därför ett utmärkt tillfälle att tillämpa och fördjupa kunskaper som förvärvats under utbildningen samtidigt som vi får möjligheter att lära oss nya tekniker, verktyg och metoder som kan gynna oss i vårt framtida arbetsliv. LUNARC har uttryckt ett behov av utökade visuella hjälpmedel för övervakning av sina beräkningskluster eftersom de nuvarande verktygen saknar intuitiv visuell representation av driftdata. Man anser därmed att det finns potential för ett gränssnitt som kan presentera periodiskt uppdaterad driftdata på en geospatial tredimensionell representation av datorhallen. Med hjälp av den föreslagna prototypen skulle det bli möjligt att undersöka potentiella samband mellan temperaturändringar, belastning, kylning etc. över tid vilket i sin tur kan leda till ytterligare optimering av framförallt systemets kylning. Det enda liknande examensarbete som vi har hittat var en undersökning av olika nätverksprotokoll inom HPC hos SAAB, men detta anser vi skiljer sig från vårt förslag eftersom vi vill undersöka en teknisk lösning för effektiv övervakning av systemets prestanda och temperaturfördelning över tid, och hur systemet modellerat i 3D skulle kunna bistå med användbara kontextuella felsökningsdata.

1.6. Avgränsningar

Inom ramen för examensarbetet ingår insamling, analys samt visuell 2D/3D-representation av loggdata från LUNARCs beräkningskluster. Loggdata sparas via LUNARCs Linuxbaserade övervakningssystem och primärt intresse för examensarbetet är data gällande temperaturer för kylsystem och serverkomponenter, aktuell belastning (arbetslast) samt inmatad effekt (W). Slutprodukten är ett webbaserat system med 3D-visualisering av loggdata kopplat till 3D-modellen som inkluderar historik.

Den utvecklade lösningen skall vara anpassad för storbildsskärm (TV) och gärna med en interaktiv styrning t.ex. via Arduino/touchpanel. Utanför de ovan nämnda områdena ingår inga

uppgifter rörande LUNARC:s infrastruktur. Examensarbetet handlar inte om att skapa av detaljerade 3D-modeller över systemet, utan kommer främst att handla om att ta fram intuitiv representation av datorhallen och de komponenter vars samverkan och placering i rummet skulle kunna bidra med.

1.7. Arbetets upplägg och fördelning

Examensarbetarna valde att arbeta agilt och tillämpade Scrum som utvecklingsmetod av flera anledningar. Det agila arbetssättet underlättade en regelbunden kommunikation mellan alla involverade parter, inklusive författarna och handledarna, vilket skapade en flexibel och dynamisk arbetsprocess där förändringar kunde hanteras effektivt samt att arbete kan delas upp i flera mindre mål. Istället för att fokusera på omfattande dokumentation och rigida processer, prioriterades samarbete och kontinuerliga iterationer, vilket möjliggjorde en mer anpassningsbar utveckling av prototypen. Scrum-metodiken tillämpades genom att arbetet delades upp i sprintar, där varje sprint innebar att en ny funktion utvecklades, testades och justerades tills den ansågs vara färdig. För att organisera arbetet användes loggböcker och issues i GitHub, där arbetsuppgifter kategoriserades i olika faser: *Backlog*, *To do*, *Doing*, *Testing* och *Done*. Denna struktur säkerställde att uppgifter kunde hanteras iterativt och att arbetes framsteg var lätt att överblicka. Kommunikation och samarbete skedde genom Microsoft Teams och Discord, vilket möjliggjorde dagliga diskussioner, snabb återkoppling och effektiv problemlösning.

	Olof Svensson	Nadjma Zaher
Analys	50%	50%
Design	50%	50%
Implementation	70%	30%
Testning/Utvärdering	30%	70%
Rapport och presentation	40%	60%
Poster	50%	50%

***Figur** 1.1:* Ungefärlig arbetsfördelning under examensarbetet.

2. Teknisk bakgrund

Detta kapitel innehåller den tekniska bakgrund som utgör grunden för de verktyg, parametrar och komponenter som förväntas simuleras av den föreslagna prototypen. Först kommer grundläggande teori, koncept och terminologi inom högpresterande datateknik att beskrivas. I det andra delkapitlet beskrivs temperaturhantering inom högpresterande datorsystem i allmänhet, och särskilt hos det system som undersöks inom just detta arbete. I det tredje delkapitlet beskrivs väsentlig terminologi och matematiska koncept inom 3D-modellering samt 3D-modelleringsbiblioteket ”three.js” och de komponenter som kommer att användas för att implementera det tredimensionella användargränssnittet.

2.1. Högpresterande datorsystem (HPD)

Högpresterande datorsystem (hädanefter HPD), är en sorts system bestående av större antal beräkningsresurser som kopplats samman till ett enda sammanhängande nätverk för att möjliggöra komplexa beräkningar och simuleringar som inte realistiskt sett kan utföras med konsumenthårdvara. Mer vardagligt brukar HPD även kallas för superdatorer. Ett sådant system består ofta av sammankopplade servrar (s.k. noder) med särskilt specialutvecklade hårdvarukomponenter ämnade åt att tillgängliggöra stora parallella beräkningsuppgifter. Även om HPD på många sätt skiljer sig från en persondator, så baseras ändå mycket av dess grundläggande utformning på samma principer. En HPD innehåller minnen för datalagring, transportmedium för dataöverföring mellan olika komponenter, input/output gränssnitt i form av skärmar och tangentbord samt specialiserade operativsystem, kompilerare, filsystem och övriga mjukvaruverktyg (Brodowicz, 2017, s.6).

Det som skiljer ett HPD från konsumentdatorer i allmänhet är just storleken på hela systemet. En konventionell dator implementerar en sorts logisk parallellism, men HPD kräver att parallellism sker både internt mellan varje nods beräkningsenheter såväl som externt mellan noder i det större systemet, vilket introducerar ytterligare nivåer av parallellism (Brodowicz, 2017, s.6-7). Noder sitter sammankopplade med varandra i grupper (s.k. kluster) där olika styrapplikationer fördelar arbetet mellan noder.

Givet är att stora system med flera högpresterande kluster inte bara kräver parallellism, utan även stora mängder energi, vilket sätter stor prägel på HPD-arkitekturen och övervakning av systemhälsa.

2.2. Övergripande arkitektur

HPD-arkitekturer bör alltid upprätthålla snabbhet och effektivitet samtidigt som energikonsumtionen inte överstiger nivåer som är fysiskt rimliga för den organisation som driver systemet (Brodowicz, 2017, s.44-46). Eftersom så stora system ofta utnyttjas av många olika användare, där kritiska beräkningar eller simuleringar står på spel, måste även systemet vara programmerbart och pålitligt nog att inte bli otillgängligt eller kompromissa viktiga data (Brodowicz, 2017, s.47).

Majoriteten av HPD-system idag (inklusive LUNARCs system) utgörs av s.k. ”commodity clusters”. Detta innebär att systemet inte köps in som en enda färdigproducerad homogen enhet, utan snarare består av konstellationer av kommersiellt specialutvecklade komponenter såsom servrar, nätverkskablar, minnesenheter och operativsystem (Brodowicz, 2017, s.77-78). Det blir alltså organisationens uppgift att få dessa komponenter att samspela på ett bra sätt, vilket i sin tur påverkar krav på felsökning och övervakning.

Kluster, är som tidigare nämnts, små nätverk av integrerade beräkningsenheter (noder) som arbetar i samspel med varandra genom att utföra parallella fördelade arbeten. Majoriteten av klusterkonstellationer idag är s.k. heterogena; d.v.s. de består av komponenter från olika tillverkare och kan ha olika utformning på den inre lågnivåarkitekturen. Detta kräver att kluster samordnas med hjälp av särskild programvara och gränssnitt. Förutom detta brukar kluster typiskt bestå av 4 komponenttyper: noder, system area network (SAM), host-noder och sekundära lagringsenheter (Brodowicz, 2017, s.85-86).

Noder utgör klustrens huvudsakliga beräkningskapacitet. Enskilda noder i kluster kan utföra isolerade arbeten, men kan även fördela arbete på andra noder inom klustret med hjälp av I/O-gränssnitt och minneshantering (Brodowicz, 2017, s.85-86). Detta sker genom ett system area network (SAM): ett nätverk av kablar där meddelanden och synkroniserade styrkommandon transporteras mellan noder (Brodowicz, 2017, s.86). Host-noden är en central styrenhet inom klustret som schemalägger och ställer om arbetsbörda och trafik efter behov (Brodowicz, 2017, s.86).

De sekundära lagringsenheterna är de lagringsminnen som innehåller alla filer, programvara, beräkningsdata osv. som behöver finnas mer långvarigt tillgängliga i klustret (Brodowicz, 2017, s.86). Denna arkitektur gör HPD-systemen modulära, skalbara, kostnadseffektiva och kraftfulla, men den heterogena uppsättningen komponenter kräver en hel del optimering och kunskap från den ägande organisationens sida, vilket i sin tur leder till att stora resurser går åt till övervakning av systemets diagnostiska parametrar (Brodowicz, 2017, s.93).

2.3. Termisk hantering i datacenter

Datacenter genererar enorma mängder värme på grund av den höga beräkningskapaciteten och den höga effekten som krävs för att driva servrar, lagringsenheter och nätverksutrustning samt den täta placeringen av servrar i rack. Effektiv termisk hantering är avgörande för att säkerställa att denna värme hanteras på ett sätt som förhindrar överhettning, som kan leda till prestandaförluster, maskinskador eller i värsta fall total systemavstängning.

Utmaningen för termisk hantering i ett datacenter är att balansera kylningens effektivitet med energiförbrukningen. Det bildas termiska ”hotspots" i vissa områden i serverhallar, vilket innebär att vissa delar av ett rack eller en server blir varmare än andra, vilket ytterligare komplicerar kylningsstrategierna. Kylsystem i datacenter använder stora mängder energi vilket har negativ miljöpåverkan och dessutom höga driftkostnader. Det finns flera olika tekniker för att hantera temperaturen i datacenter, var och en med sina specifika fördelar och utmaningar (Alkharabsheh et al., 2015).

2.3.1. Luftkylning

Den traditionella och mest använda metoden är luftkylning, vilket fungerar genom att cirkulera kall luft runt serverrack och noder och ventilera ut den varma luften. Det finns olika typer av luftkylningssytem (Fulpagare & Bhargav, 2014).

Det finns en luftkonditioneringsteknik s.k. CRAC (Computer Room Air Conditioners), där luften kyls ned av ett internt köldmedium och cirkuleras i rummet. CRAC-enheter används oftast i mindre datacenter för att cirkulera luften. CARC-enheterna kontrollerar luftflödet och övervakar temperaturen, luftflödet och fuktigheten i datacentret. En av nackdelarna med CRAC-enheterna är att de kan vara mindre effektiva när det gäller att rikta kall luft där det behövs vilket resulterar i energislöseri. En annan teknik är CRAH (Computer Room Air Handling System), vilket liknar CRAC men här används ett indirekt kylsystem där luft kyls av vatten istället för ett köldmedium. CRAH-enheterna används i större datacenter där man vill maximera kylningseffektivitet.

Ytterligare en teknik är där man använder kall- och varmgångarna (hot aisle/cold aisle containment) för att optimera luftflödet och förhindra att kall och varm luft blandas. Det sker på så sätt att kall luft blåses genom rader av servrar, medan den varma luften från servrarna leds bort genom separata varmgångar vilket hjälper till att minimera temperaturökningar och maximerar effektiviteten i kylningen (Alkharabsheh et al., 2015).

Luftkylning anses vara enkel att implementera men har begränsningar när det gäller effektivitet vid kylning av tätt packade rack eller stora datacenter. Luftkylda datacenter tenderar att utveckla “hotspots” i områden där värme inte effektivt sprids bort vilket kan leda till överhettning. För att hantera detta används ofta lösningar såsom sensorer för att identifiera hotspots, dynamisk luftflödesstyrning eller kylda golv där kall luft blåses upp genom golvplattor direkt mot servrarna. (Fulpagare & Bhargav, 2014).

2.3.2. Vattenkylning

Även om luftkylning är den vanligaste kylningsstrategi i datacenter, så är det inte tillräckligt effektivt på grund av luftens låga densitet och värmeavlägsningskapacitet. Vattenkylning är en mer avancerad metod för att hantera värmebelastning i datacenter. Metoden blir allt vanligare eftersom det erbjuder högre effektivitet och bättre prestanda i att avleda värme jämfört med luftkyla. I denna metod cirkuleras vatten genom kylrör antingen direkt eller nära på komponenter för att absorbera och transportera bort värme (Zhang et al., 2018).

I direkt vattenkylning cirkuleras vatten direkt till servrarnas komponenter via kylblock som är monterade vid processorer och andra ställen där det genereras värme. Detta sker med hjälp av dielektriska vätskor som ger elektrisk isolering. Detta tillvägagångssätt kan vara mycket effektivt och säkerställer en effektiv värmeavledning eftersom vatten har en högre värmeledningsförmåga än luft.

Med indirekt vattenkylning kyls luften genom att använda vatten som kylmedium utan att det direkt kommer i kontakt med komponenterna. Värmen överförs från servrarna till vatten vanligtvis genom användning av vattenkylda dörrar som monteras på baksidan av rack. Dörrarna innehåller vätskebaserade värmeväxlare som tar värmen som genereras av servrarna och överför till vatten som sedan transporteras och kyls ned i en central kylenhet.

Vattenkylning erbjuder en effektiv lösning för dagens högpresterande datacenter eftersom det reducerar energiförbrukningen pga. det minskade behovet av att ha fläktar samt vattnets förmåga att absorbera mer värme än luft. Det möjliggör även att man kan ha tätt packade rack i ett mindre utrymme med mindre risk för överhettning pga den effektivare värmehantering som vattenkylningen erbjuder. Samtidigt som det finns fördelar med denna metod så finns det risk för läckage och därför krävs det underhåll för att förhindra eventuella läckor som kan skada komponenter. Dessutom är Initialkostnaden för installation och underhåll av vattenkysystem högre än för luftkylningssytem (Khalaj & Halgamuge, 2017).

2.3.3. Hybridkylning

I många moderna datacenter kombineras fördelarna med både luft- och vattenkylning för att hantera de termiska förhållandena i ett datacenter och maximera effektiviteten. Ett hybridkylsystem kan utnyttja luftkyla för att hantera grundläggande kylbehov, medan vattenkyla används för att hantera specifika områden med hög värmebelastning. Genom att placera luft-till-vätska-kylaren i närheten av racken så kommer varm luft kylas ned direkt istället för att varmluft ska cirkulera i systemet och belasta CRAC-enheterna. Fördelarna med denna lösning är att det minskar behovet av luftkylning vilket innebär lägre energiförbrukning. Nackdelen är läckage samt att det kan kräva noggrann underhållning och övervakning (Alkharabsheh et al., 2015).

2.3.4. Free cooling

Free cooling eller frikyla innebär att man utnyttjar naturlig källa för att kyla datacenter. I områden med kallt klimat kan frikyla minska energiförbrukningen genom att använda utomhusluft för att kyla servrarna. Eftersom den här metoden reducerar behovet av mekanisk kylning så innebär det även en minskad underhållskostnad samt minskad miljöpåverkan. Begränsningarna med denna metod är att det kräver kall klimat året runt för att det ska funka samt att utomhusluftens fuktighet kan i vissa fall vara skadlig för komponenterna. (Zhang et al., 2018).

2.4. 3D-grafikens grunder

3D-grafik är idag vanligt förekommande och de allra flesta har någon sorts erfarenhet av 3D-grafik i vardagen. Vanligen möter människor 3D-grafik i form av realtidsrendering i spel och modelleringsapplikationer eller förrenderad grafik inom film och tv, så det grundläggande konceptet i sig kräver ingen vidare introduktion. För att beskriva 3D-grafik är det värdefullt att ha en förståelse kring de särskilda termer, koncept och matematiska modeller som utgör grunden för tekniken. För att ge en inblick i de tekniker som implementeras inom detta projekt, kommer detta avsnitt handla om grundläggande koncept inom realtidsrenderad 3D-modellering såväl som designprinciper rörande 3D-baserade gränssnitt.

De flesta realtidsrenderade 3D-objekt består av polygoner: en sorts strukturell tvådimensionell byggsten som ger objektet sin form. Polygoner bygger i grund och botten på enkel geometri och består av ytor, kanter och hörn varav hörnen representeras internt av matriser som kontinuerligt beräknas om baserat på objektets position, rotation, storlek, eller den virtuella kamerans placering i 3D-rummet (Gahan, 2011, pp.21-44). Typiskt utgörs realtidsrenderade 3D-modeller av trianglar för att hålla matriser så beräkningsmässigt billiga som möjligt (Gahan, 2011, pp.21-44).

Vanligtvis kallas hela objektets polygonstruktur för dess "geometri", då den i detta skede inte ännu består av någon egentlig grafik utan bara teoretiska punkter i x- ,y- ,z-planet. För att 3D-objekt ska renderas och synas på skärmen måste objektet även ha en s.k. "mesh", alltså färgpixlar som simulerar en yta som reflekterar ljus (Gahan, 2011, pp.21-44). En mesh gör objektet synligt, men enbart solida färger gör att objektet inte upplevs som fullt tredimensionellt, utan mer som en färgad siluett. Därför tillsätts "shaders": algoritmer som beräknar vilka punkter på objektets mesh som ska ha en mörkare färgton och vilka som blir ljusare, baserat på var objektet befinner sig i relation till simulerade ljuskällor (Gahan, 2011, 139-140). Shaders behöver inte bara representera skuggning av ljuskällor, utan kan också manipuleras manuellt efter anda parametrar för att efterlikna t.ex. heatmaps.

2.4.1. 3D-gränssnitt

Ett av projektets delmål är att det ska skapas ett interaktivt, dynamiskt 3D-användargränssnitt där viktig driftinformation kan presenteras spatialt. Introduktionen av en z-axel påverkar interaktionen med datarepresentationer, vilket i sin tur ställer särskilda krav på designen av just 3D-gränssnittet (Molina & Gonzalés, 2009, pp.103-104). En stor utmaning som uppstår inom 3D-design är att all information inte kan presenteras effektivt med bara tredimensionella modeller, utan det krävs ofta ett samspel mellan 2D-element och 3D-element, vilket gör det nödvändigt att se till att olika element kompletterar varandras styrkor och svagheter (Molina & Gonzalés, 2009, pp.103-104). Dessutom påverkar en användares erfarenheter med ett visst gränssnitt användarupplevelsen, så om en användare inte är särskilt van vid navigering inom virtuella tredimensionella rum ställs ytterligare krav på designen (Johnson, 2020).

Trots en hel del olikheter finns det likheter mellan 3D- och 2D-gränssnitt. På samma sätt som användare kan navigera mellan olika vyer i 2D, kan en användare hoppa mellan synvinklar och positioner i 3D-rummet som presenteras dynamiskt i realtid som 2D-bilder på skärmen (Molina & Gonzalés, 2009, pp.105-108). Man kan tänka sig detta som om varje rörelse med musen får en 3D-kamera att röra sig i virtuella rummet, vilket triggar en övergång från en vy till en annan precis som om man hade bläddrat mellan vyer i ett 2D-baserat gränssnitt.

Mer betydande är de många skillnader som finns mellan 2D och 3D-gränssnitt. Den tydligaste skillnaden är att 2D-gränsnitt förhåller sig till utrymmet som pixlar i x- och y-planet där olika konstellationer av grafiska komponenter existerar avgränsade från varandra, medan komponenter i 3D-rum kan upplevas som sammankopplade i ett kontinuerligt utrymme och skyms och visas enligt den vituella ”kameran” innuti scenen (Molina & Gonzalés, 2009, pp.105-108). Med andra ord kan man beskriva det som att användaren kan ”flyga” oavbrutet från en 3D-knapp till en annan om man så vill, medan 2D knappar antingen renderas tillsammans eller i olika vyer vilket kräver att användaren bläddrar mellan vyer. Molina och Gonzales (2009) beskriver att det finns två huvudsakliga sätt att kombinera 2D-element i 3D-rum: den ena är i form av fönstervyer som placeras ut med koordinater i rummet och den andra är i form av statiska 2D element som alltid finns i användarens synvinkel: s.k. ”Head Up Displays” (HUD).

2.4.2. Three.js (WebGL) och 3D-grafik i JavaScript-applikationer

WebGL är ett bibliotek med öppen källkod utvecklat av Khronos Group för att möjliggöra realtidsrendering av 3D-grafik direkt i webbläsare (Khronos Group, 2024). Förutom eliminering av tredje-parts plugins för rendering utvecklades WebGL för att tillåta kompatibilitet med HTML5, portabilitet till olika plattformar och större lågnivå-kontroll till utvecklare (Khronos Group, 2024). Ett resultat av WebGL:s arbete är JavaScript-biblioteket Three.js som har syftet att förenkla integrering av 3D-grafik i JavaScript-baserade webb-applikationer samtidigt som WebGL:s öppna källkods-princip fortfarande stöds (Cabello, 2024).

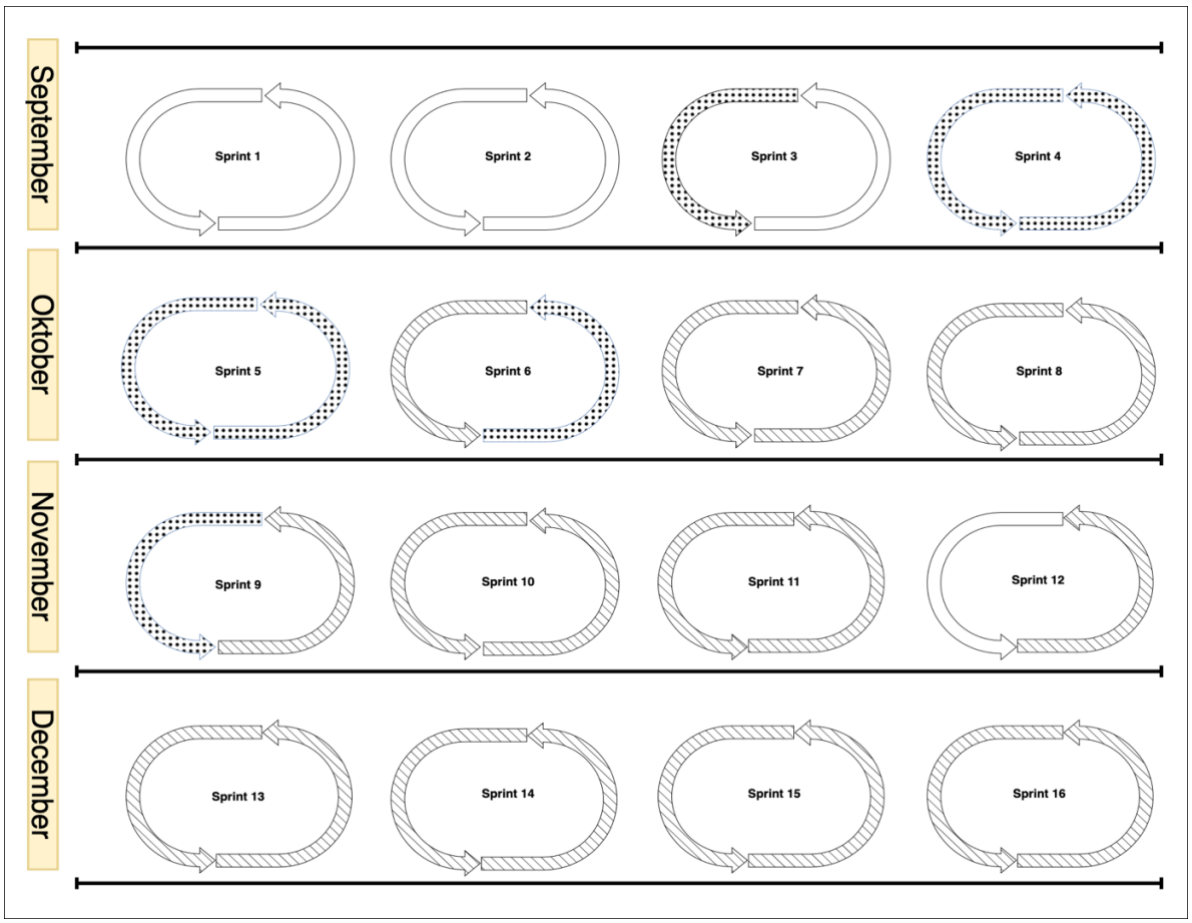
Three.js förenklar implementering av 3D-grafik tack vare att det innehåller redan fördefinierade komponenter som kan användas direkt i JavaScript-projekt. Man kan se dessa komponenter som objekt som vid initialisering kan ställas in med hjälp av olika parametervärden. Förutom komponenter som kameror, ljuskällor och scener, har Three.js även komponenter som representerar enkelt konfigurerbara former av olika typer. För att rendera ett objekt på skärmen behöver man initialisera kamera och ljus, vilket tillsammans skapar scenen. Sedan lägger man till ett objekt av godtycklig form: t.ex. ett rätblock. Sedan behöver man endast ge objektet ett variabelnamn och skapa en ny instans av objektet med en geometrityp och en materialtyp från alla de komponenter som redan finns färdigutvecklade av Cabello för Three.js (Cabello, 2024). Sist behövs en renderare, alltså den komponent som tar hand om alla matrisberäkningar, konverterar dem till 2D-bilder med simulerat djup och ljussättning och skickar dem till den s.k. frame-buffern: en sorts buffer i minnet som laddas i olika frekvens för att simulera rörliga bilder i realtid (Cabello, 2024).

3. Metod

I detta kapitel beskrivs hur arbetet har genomförts, från den initiala planeringen till slutanalysen och rapportskrivningen. Det redogörs för de olika faserna i arbetsprocessen, hur resultaten från en fas har påverkat arbetet i de följande faserna och de metoder och verktyg som har använts för att samla in och analysera information och data. Dessutom motiveras metodval samt beskrivs hur kommunikationen skett internt och med LUNARC.

3.1. Arbetsprocessens faser

Examensarbetet delades upp i veckovisa sprints i faser enligt figur 3.1 nedan och innehöll: efterforskningsfasen, kravhanteringsfasen och utvecklingsfasen. Varje sprint i processen kunde delas upp i planering, implementering och utvärdering.



Figur 3.1: Helfärgat indikerar efterforskning, prickar indikerar kravhantering och ränder indikerar utveckling. Notera att sprintar kunde innehålla element av olika sorters arbete vid särskilda skeden.

3.1.1. Efterforskning

I efterforskning ingick att samla in relevant information och litteratur inom ämnesområdet. Det utfördes en omfattande litteraturstudie för att identifiera aktuella metoder och tekniker som används inom området. Som en del av efterforskningen studerades även 3D-modellering samt 3D-modelleringsbiblioteket three.js för att få kännedom om JavaScript och dess tillämpning inom vårt projekt. Genom att granska vetenskapliga artiklar, rapporter och litteratur fick examensarbetarna en grund att bygga vidare på. Det användes databaser som Google Scholar och IEEE Xplore för att hitta relevant forskning. Resultatet av denna fas användes för att formulera en tydlig problemställning och syfte för arbetet samt ett underlag för eliciteringen inför nästa fas. Intervjuer och möten dokumenterades med hjälp av mötesprotokoll där viktig information och olika insikter sparades undan för framtida analys och diskussion.

3.1.2. Kravhantering

I kravhanteringsfasen planerades och tillämpades olika eliciteringstekniker såsom intervjuer och observationer för att säkerställa en omfattande och otvetydig kravspecifikation. Efter inledande möten som hölls med LUNARC, diskuterades och definierades krav enligt intressenternas önskemål. Examensarbetarna observerade även deras arbetsmiljö och serverhallen, vilket möjliggjorde identifiering av behov och utmaningar med befintliga system. Examensarbetarna strukturerade och prioriterade kraven med MoSCoW-metoden, där kraven kategoriserades i fyra grupper: Must have, Should have, Could have och Won't have this time. Detta tillvägagångssätt säkerställde att fokus lades på de mest kritiska kraven först. Dessa krav dokumenterades i en kravspecifikation som senare blev vägledande för utvecklingsfasen.

3.1.3. Utveckling

Utvecklingsfasen genomfördes iterativt och agilt enligt en förenklad variant av scrum. Varje vecka hölls minst två möten: ett möte i början av veckan där den kommande sprinten planerades och en i slutet av veckan där sprinten utvärderades och diskuterades. Fördelningen av arbetsuppgifter hanterades med hjälp av GitHub:s “issues”-system, där man kan skapa arbetsuppgifter som planeras hända enligt kravspecifikation, såväl som spontant under utvecklingens gång. Issues kunde sedan läggas in i en SCRUM-board för bättre överblick och hantering av arbetsuppgifter.

Varje gång ändringar gjordes, skrevs det kommentarer i den relevanta uppgiftens issue för att förtydliga vad som hade gjorts och varför uppgiften ansågs ha blivit färdigställd. Projektet avser skapa tre huvudsakliga komponenter: en frontendapplikation, en express.js-server som levererar applikationen till klienten och ett API som periodiskt läser av indata i CSV-format. Detta innebar att vi delade in projektet i två separata GitHub-projekt för att hålla frontend- och backendfunktionalitet tydligt separerade under projektets gång.

Frontend-applikationen utvecklades med hjälp av React: ett JavaScript-bibliotek som underlättar utveckling av interaktiva komponenter (React, 2024). Biblioteket Three.js för att skapa ett interaktivt och visuellt 3D-gränssnitt som representerar data från serverhallen olika noder och rack. Frontend-applikationens struktur utgick från olika komponenter, där varje komponent ansvarade för specifika funktioner och visualisering av data. Den primära användarupplevelsen bygger på en interaktiv 3D-vy skapad med Three.js där rack och noder representeras visuellt. Dessa komponenter använder specifika texturer och färger som anpassas efter rackens och noders temperatur, vilket ger användaren en tydlig visuell indikation på exempelvis överhettade noder. Det genereras även gradientfärgade texturer beroende på temperaturdata. För datahämtning och API anrop från backend-servern används Axios: ett JavaScript API som förenklar formateringen av URL:er till backend endpoints(Axios, 2024). Applikationen kan anropa specifika routes för att hämta information om det giltiga tidsintervallet för nytillkommen data, racknamn och tillhörande noder, noders positioner i racken, samt en nods parametervärden för ett specifikt tidsintervall eller alla parametervärden för alla noder under en enda tidpunkt.

I 2D-gränssnittet användes Chart.js: ett JavaScript-bibliotek som möjliggör flexibel implementering av dynamiska grafer i JavaScript-projekt (Chart.js, 2024). Detta var för att visa detaljerade grafer över temperatur, energiförbrukning och arbetsbelastning för noder i applikationen. Användaren kan anpassa visningsstorleken på grafen och dra fönstret till önskad position. Användaren kan även navigera och integrera med applikationen genom en kontrollpanel där de kan styra applikationens olika vyer och grafinställningar, samt få tillgång till insamlade data för enskilda noder eller rack. Detta gränssnitt förbättrar användbarheten genom att göra det enkelt att växla mellan vyer och analysera specifika dataelement.

Prototypens backend-lösning utvecklades med hjälp av Node.js och ramverket Express.js. Den här servern hanterade inläsning och bearbetning och leverans av data som behövdes för applikationen, med fokus på att interagera med noder och rack i en datamiljö. Backend-servern konfigurerades för att kunna läsa in två CSV-filer, nodelist.csv och readings.csv som innehåller information om enskilda noder och deras aktuella läsningar. Dessa filer omvandlades periodiskt till ett JSON-format vilket förenklades och snabbade upp dataleverans till frontend. Genom att använda biblioteket PapaParse kunde servern enkelt konvertera CSV-innehåll till JSON. Data från båda CSV-filerna kombinerades för att ge en detaljerad bild av varje nod, inklusive information om temperatur och energiförbrukning.

Systemet uppdateras med ett tidsintervall på 10 minuter där CSV-filerna läses in. För att tillgängliggöra den bearbetade datan skapades olika API-routes för att returnera information om alla noder i ett specifikt rack och information om en specifik nod baserat på dess alias. För att möjliggöra användning och göra säker dataöverföring mellan frontend och backend aktiverades CORS (Cross-Origin Resource Sharing) på servern.

3.1.4. Testning

Tidigt i projektet specificerades ett antal testtyper för prototypen: enhetstester, funktionstester, systemtester och användartester. Huvudsakligen utfördes enhetstester och funktionstester genomgående i samband med integrering av nya uppdateringar till GitHub-repot. Det skapades även ett antal automatiska tester för att se till att särskilda prestanda- och dataformat-krav kunde verifieras. På grund av den stora mängden interaktiva komponenter i 3D- och 2D-gränssnitten så bestod en större del av funktionstesterna i slutändan av manuella tester. Den modulära designen i kombination med React-bibliotekets olika tillståndsfunktioner gjorde att mer fokus kunde riktas åt just direkt interaktion med systemet och stresstester för datahanteringen, snarare än automatiska tester av separata komponenters olika tillstånd.

Inför varje uppdatering hölls ett projektmöte där de nytillkomna funktionerna diskuterades, testades och utvärderades för att säkerställa att de inte påverkar tidigare funktionalitet. På grund av logistiska begränsningar hos LUNARC så beslutades det att användartester kunde begränsas till ett antal demo-presentationer tillsammans med intressenter från LUNARC där prototypens alla ingående delar kunde diskuteras i detalj. Dessa diskussioner skulle antecknas, formuleras om och föras in i GitHub som “issues”, eller nya arbetsuppgifter i projektets olika uppgifter.

Det utvecklades en uppsättning tester för att verifiera backendens stabilitet och korrekthet. Syftet var att säkerställa att data hämtas och bearbetas som förväntat, samt att API:et reagerar på ett pålitligt sätt under olika omständigheter. Ett av målen var att bekräfta att backend-servern kunde läsa in och tolka JSON-filer utan problem. Till detta ändamål skapades tester som kontrollerade att en JSON-fil lästes korrekt, samt att felaktigt formaterad data hanterades genom tydliga felmeddelanden. Dessutom testades omvandlingen från CSV- till JSON-format, för att se till att systemet kunde tolka inkommande data och presentera den i ett enhetligt och användbart format. Testerna kontrollerade både att konverteringen fungerade, och att resultatet sedan sparades på ett korrekt sätt.

När det gällde API:et som användes för att hämta rack- och nodinformation, testades olika scenarion. Exempelvis verifierades att rätt data returnerades för ett befintligt rack eller nod, samt att ett lämpligt felmeddelande och statuskod gavs om efterfrågad data saknades. På så sätt säkerställdes att en tydlig återkoppling gavs till användaren, oavsett om data fanns tillgänglig eller ej.

Slutligen utfördes även enklare belastningstester, där flera samtidiga förfrågningar skickades till servern. Detta gjordes för att se om systemet klarade att hantera hög efterfrågan utan att prestera sämre eller ge felaktiga svar. Resultatet gav en indikation på prototypens skalbarhet och stabilitet.

3.1.5. Utvärdering

I samband med varje veckovisa sprint-retrospektiv förekom diskussioner mellan examensarbetarna om vad som hade gjorts under veckan, hur det hade gått, vad som hade varit svårt eller enkelt och vilka buggar eller problem som hade upptäckts i samband med testningen och utvecklingen av nya funktioner. Kod-diskussioner förekom också där examensarbetarna satt med koden och diskuterade problem som hade dykt upp som krävde särskilt fokus. Detta blev ofta en bra möjlighet att repetera saker som man hade lärt sig under examensarbetets gång, såväl som ett sätt att dela med sig av information som framöver kunde gynna examensarbetet. Tester som utfördes i samband med färdigställandet av nya funktioner skedde alltid innan integrering med GitHub. En regel sattes där funktioner inte kunde laddas upp utan att den på något sätt hade testats, presenterats och diskuterats på ett sprint-möte.

3.1.6. Examensarbetesrapport

Rapportskrivandet löpte parallellt med utvecklingen som en dokumentation av processen. En stor del av rapportskrivandet tog även stöd av alla de anteckningar som hade förts ner i examensarbetets gemensamma loggbok. Tanken var att alla de insikter, funderingar och övriga detaljer från möten skulle sparas i loggboken för att användas i rapportens analys- och resultatkapitel. Till hands fanns även kod-kommentarer, information kring issues i GitHub och anteckningar från möten med LUNARC, vilka användes flitigt vid skapandet av rapporten.

3.2. Källkritik

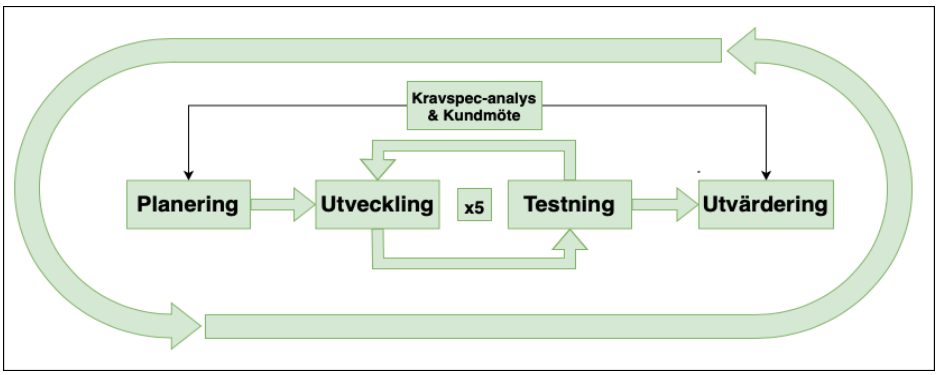
Examensarbetet ska baseras på vetenskaplig grund där informationen är tillförlitlig och relevant och stöttas av legitima källor. Således är det av största vikt att samtliga källor utvärderas för sin tillförlitlighet. Källförteckningen består av tryckt litteratur och vetenskapliga artiklar och dokumentation.

Tryckt litteratur och vetenskapliga artiklar anses tillförlitliga, dels eftersom författarna till litteraturen anses vara experter inom sina respektive områden, dels för att innehållet i texterna granskas före publicering. Det gäller för källorna (Alkharabsheh et al., 2015), (Fulpagare & Bhargav, 2014), (Khalaj & Halgamuge, 2017) och (Zhang et al., 2018) som är artiklar publicerade i referentgranskade vetenskapliga tidskrifter.

Även källorna (Johnson, 2020), (Molina & Gonzalés, 2009) och (Brodowicz, 2017,) är litteratur och bokkapitel utgivna av välrenommerade förlag. Böckerna har genomgått redaktionell process vilket säkerställer att innehållet är vetenskapligt underbyggd. Dessa källor erbjuder omfattande teoretisk och praktisk kunskap som är värdefull för att förstå de tekniska aspekterna i detta arbete. Källor som består av dokumentationer och teknisk beskrivning är den officiella dokumentationen för Three.js och WebGL vilket underhålls av skaparen av biblioteket vilket garanterar att informationen är korrekt, aktuell och direkt från den primära källan. Dessa källor anses vara pålitliga eftersom företag inte får bedriva falsk marknadsföring. Detta gäller för källorna (Cabello, 2024) och (Khronos Group, 2024).

4. Analys

I detta kapitel analyseras de centrala delarna av arbetets utveckling och de beslut som fattats under arbetets gång. Fokus ligger på de olika komponenterna i systemet – 3D- och 2D-gränssnittet och hur datahanteringen och integrationen mellan frontend och backend – utformades och optimerades. Vidare diskuteras de utmaningar som identifierades under utvecklingen, samt hur dessa hanterades för att säkerställa en funktionell och effektiv prototyp. Analysen belyser också de tekniska beslut som påverkat systemets slutliga form, inklusive hantering av texturer, visualisering av temperaturdata samt implementeringen av varningssystemet. Följande avsnitt går djupare in på de viktigaste aspekterna av utvecklingsarbetet och hur dessa har bidragit till att uppfylla projektets mål.



Figur 4.1: En mer detaljerad bild av hur en sprint i utvecklingsprocessen såg ut, samt de moment som kontinuerligt bidrog till problemlösning och beslutsfattande.

4.1. Kravspecifikationen

Arbetet med kravspecifikationen bestod av fyra huvudsakliga moment: elicitering, analys, verifiering och prioritering. Elicitering av krav påbörjades direkt i projektets inledande fas, där anteckningar från tidigare möten med LUNARC sammanställdes. Identifiering av intressenter, domäner, arbetsområden som innefattas av projektet, samt de metoder som kommer att användas för identifiering av krav inom särskilda områden. Denna tidiga process gjorde det enklare att fastställa vad som ännu var okänt kring examensarbetets delmål och vad som mer säkert kunde fastslås. I slutet av den inledande eliciteringsfasen hölls ett möte med LUNARC där examensarbetarna fick möjlighet att ställa alla de frågor som hade dykt upp under processens gång.

Kravspecifikationen utgjorde grunden i detta arbete och definierade de funktionella och icke-funktionella krav som styrde utvecklingen av prototypen som övervakar och visualiserar serverhallens termiska förhållanden. Med fokus på både 3D- och 2D-gränssnitt, samt effektiv datahantering och användbarhet, formulerades kraven för att uppnå en balanserad och praktiskt användbar lösning. En fullständig kravspecifikation finns tillgänglig i rapportens appendix.

Ett av de mest kritiska funktionella kraven var att utveckla ett interaktivt 3D-gränssnitt som återspeglar serverhallens fysiska miljö, med rack och noder positionerade som i serverhallen. Gränssnittet behövde uppdateras dynamiskt baserat på temperaturdata och visuellt representera dessa variationer med hjälp av färgskalor, vilket möjliggjorde identifiering av potentiella problemområden såsom överhettning. Denna funktionalitet var avgörande för att ge användarna en tydlig överblick över hallens status.

För att komplettera 3D-visualiseringen specificerades ett 2D-gränssnitt som skulle presentera loggdata i form av grafer. Dessa grafer behövde visa diagnostiska parametrar som temperatur, energiförbrukning och arbetsbelastning på ett sätt som underlättade analys av långsiktiga trender. Vidare behövde gränssnittet innehålla ett tidsreglage, vilket gjorde det möjligt för användaren att navigera genom historiska data och analysera specifika ögonblick. En viktig del av specifikationen var integrationen mellan 3D- och 2D-gränssnitten. När en användare valde ett rack eller en nod i 3D-modellen skulle denna handling trigga uppdateringar i 2D-gränssnittet, exempelvis visning av detaljerade grafer för den valda komponenten.

Prestanda och skalbarhet var centrala icke-funktionella krav. Prototypen behövde hantera realtidsuppdateringar av data med hög responsivitet och samtidigt behålla en bildhastighet på minst 30 bilder per sekund. Det var också viktigt att systemet kunde skalas för att hantera framtida utbyggnad av serverhallen och att det designades för att enkelt kunna integreras med ytterligare API:er för direkt hantering av LUNARC:s interna loggdata. Användbarhet var en annan nyckelkomponent. Vid felaktig data eller andra problem skulle systemet visa tydliga och koncisa felmeddelanden för att förhindra förvirring och säkerställa att systemet ska förbli responsivt och funktionellt, så att användaren kan fortsätta navigera och analysera tillgänglig data även om vissa delar saknas eller är felaktiga.

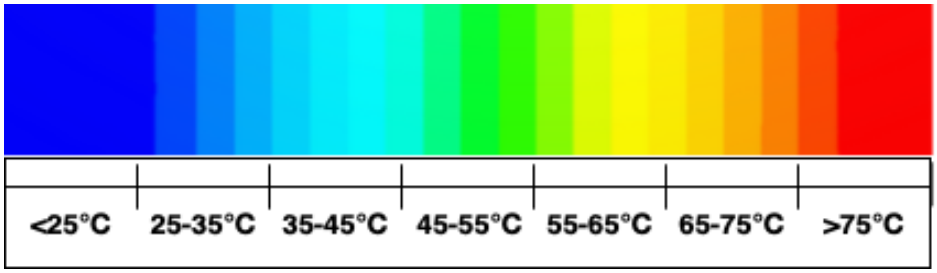
4.2. 3D-gränssnittet

Under första delen av 3D-gränssnittets utveckling fokuserades på att definiera funktioner som returnerade renderade meshes av rack och noder. Dessa krävde initialt endast parametrar som specificerade komponentens position i 3D-rummet, men utvecklades så att ytterligare in-parametrar kunde hanteras framöver utan att funktionen slutar fungera. Ett beslut fattades där noder och rack skulle existera som egna komponenter samtidigt som rack även kunde fungera som en fabrik för noder om man väljer att skicka med ett nod-antal som in-parameter. Detta kunde göras utan att de båda klasserna hade någon koppling till varandra, d.v.s. rack kan instansernas utan noder, och noder kan instansernas utan rack. En stor anledning till att detta ansågs effektivt var just att noder måste tilldelas samma x- och z- koordinater som rack när de placeras inuti dem. Därför ansågs det lämpligt att låta noder ritas ut vid skapandet av rack för att göra huvudapplikationens kod mer fokuserad och läsbar.

Efter att 3D-noder och rack hade placerats ut lades fokus på interaktion med objekten. Till en början handlade det om att man skulle kunna klicka på ett objekt i rummet, för att detta sedan skulle returnera ett unikt id som tilldelats objektet när det skapades. Detta kunde uppnås med hjälp av s.k. “event-listeners” som vid varje klick triggade en funktion som sorterar alla 3D-objekt-referenser som skärs av en stråle som går från det klickade området och genom hela 3D-rummet. Man kunde då få komponenten som var först i listan att skicka sitt id-nummer till konsolen, vilket sedan kunde användas vid felsökning av mer avancerad funktionalitet.

Med både 3D-objekt och interaktion med objekten på plats, var det dags att få temperaturskalor att ritas ut över komponenterna. Detta gjordes initialt endast på rack, eftersom de bestod av samma sorts geometri som noder. Med andra ord kunde samma lösning återanvändas även på nod-komponenter. Till en början uppstod det problem med renderingen av en enda enhetlig gradient-textur som fästes på box-geometrin. Tanken var att texturen skulle fungera som att man hade spänt en mönstrad filt över lådan, där mönstret inte bryts och renderas om vid varje hörn. I och med detta gjordes felbedömningen att texturer fick placeras på individuella av meshes av höger- och vänster sida, ovansidan och undersidan, samt framsidan och baksidan på noden.

För ökad prestanda sänktes texturupplösningen och det lades även till en sorts textur-buffert där nya texturer kunde lagras för återanvändning. På så sätt kunde man se till att texturerna inte beräknades om i samma takt som bildfrekvensen, dvs.60 om-beräkningar per sekund gick ner till minst en beräkning per 10:e minut. Denna lösning visade sig dock varken ge tydliga och läsbara gradientskillnader eller uppfylla prestandakrav för applikationen, så till följd av ett möte med LUNARC bestämdes det att dessa texturer behövde omarbetas.



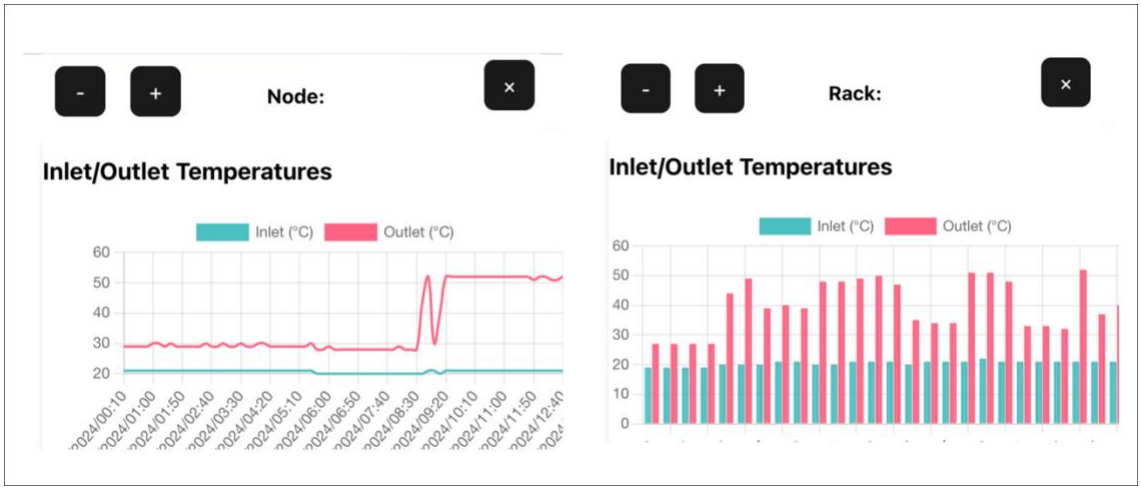
Figur 4.2: Den temperaturgradientens färgskala och motsvarande temperaturintervall för varje färgövergång.

Under utvecklingen av 3D-gränssnittet identifierades att tidigare implementation av texturer och gradientfärger inte fullt ut speglade den korrekta färgskalan. För att förbättra färgskalan på de renderade objekten krävdes en omarbetning av hur texturer och gradienter skapades och mappades till varje nod och rack. Den tidigare lösningen visade sig vara alldeles för ineffektiv, men tack vare bra tips från LUNARC gick det att ändra strategin för hur texturerna beräknades och placerades över noderna. Ett av dessa råd var att: i stället för att arbeta med nodernas texturer nod för nod, så kunde man ta nod-geometrins hörnkoordinater och placera en enda gradient över alla noder i racket samtidigt. Detta förminskade texturberäkningarna markant och färgövergångar kunde göras mer enhetliga, vilket resulterade i en mer korrekt visuell representation av de termiska förhållandena. Som följd av detta skapade de nya texturerna en mer realistisk och informativ visualisering av temperaturdata (se figur 4.2).

I samband med denna omarbetning av texturer infördes nya komponenter som behöll samma grundläggande funktionalitet som tidigare, men som är utökade med stöd för mer högupplösta, noggrannare färgskalor. De nya texturerna krävde en analys av färgskalan och de färger på skalan som skulle motsvara ett temperaturvärde. Genom diskussioner med LUNARC utifrån den temperaturdata som fanns tillgänglig bestämdes det att färgskalan skulle se ut enligt figur 4.2. Temperaturgradienten går mellan 25–75°C där vädren under 25°C indikeras av blå färg och värden över 75°C indikeras av röd färg. Temperaturvärden inuti intervallet ansågs vara typiska för LUNARC:s noder i drift, så dessa bedömdes vara rimliga att koppla till färger mellan blått och rött i färgskalan (se figur 4.2). Utöver detta inkluderades en ny varningsfunktion som presenterar varningar i små fönster, där varje berörd nod tydligt kan kopplas till varningen via en röd linje som ritas från noden till varningsrutan. På så sätt kunde användaren snabbt identifiera problemområden, till exempel noder med onormalt hög temperatur eller ovanlig kombination av effektförbrukning och CPU-belastning. Detta fönster innehöll information om nodens tillstånd, inklusive temperaturer, CPU-belastning och energiförbrukning. Sammantaget ökade dessa uppdateringar tydligheten och användbarheten i gränssnittet, samtidigt som systemets prestanda, flexibilitet och informationsförmedling förbättrades.

4.3. 2D-gränsnittet

2D-gränssnittet designades för att presentera data i tydliga och lättanvända grafer som gav användaren en snabb överblick över temperatur, energiförbrukning och arbetsbelastning i serverhallen. Chart.js användes som huvudsakligt verktyg för att skapa dessa grafer, vilket erbjöd ett flexibelt och kraftfullt sätt att hantera dynamiska data som hämtades från backend-servern. Det innebar att grafens innehåll kunde uppdateras i realtid eller med ett specifikt intervall, vilket var särskilt viktigt för att visa fluktuerande värden som temperatur och energiförbrukning, Genom att välja en diagramlayout kunde man representera dessa variabler över tid på ett intuitivt sätt, vilket underlättade användarens analys av långsiktiga trender och avvikelser i systemets prestanda.



Figur 4.3: Graferna skulle visa antingen en dags parametervärden för en nod, eller alla noders ögonblicksvärde för ett givet rack.

Utvecklingen inkluderade dessutom en flyttbar och resizable grafkomponent, vilket var tänkt att förbättra gränssnittets flexibilitet och anpassning. Genom att låta användaren dra och ändra storlek på grafen kunde gränssnittet anpassas till individuella behov och förhindra att grafen blockerade 3D-vyn. Den här flexibiliteten förbättrade arbetsflödet eftersom användaren enkelt kunde växla mellan detaljerad dataanalys och 3D-navigering utan att lämna applikationen. En kontrollpanel integrerades också i 2D-gränssnittet för att låta användaren direkt växla mellan rack- och nod-grafer. Kontrollpanelen möjliggjorde även snabb åtkomst till uppdaterade data om noder och rack, vilket var tänkt att göra det enklare att djupdyka i specifika problemområden.

För att säkerställa att 2D-gränssnittet bidrog till en positiv användarupplevelse låg fokuset på att göra det lättnavigerat och intuitivt. Genom att organisera data i tydligt definierade sektioner – en graf för temperaturer, en för energiförbrukning och en för arbetsbelastning – kunde användaren snabbt identifiera viktiga mätvärden och analysera specifika parametrar.

En av de primära utmaningarna var att få grafkomponenten att renderas och interagera med huvudapplikationen utan att störa andra funktioner, som 3D-gränssnittet. Examensarbetarna behövde finna en möjlig lösning på hur grafen kunde uppdateras och visas utan att blockera eller fördröja interaktionen med andra delar av applikationen. Detta krävde att komponentens struktur inte påverkade prestanda alltför negativt.

Under utvecklingsprocessen implementerades flera förbättringar i grafkomponenten för att öka användbarheten och funktionaliteten i 2D-gränssnittet. En av de mest betydande ändringarna var införandet separata grafkomponenter för rack och noder. Det innebär att man nu kan öppna och visa grafer för ett rack och en nod samtidigt utan att de behöver dela på samma graf-fönster. Genom att använda tillståndsvariabler som isGraphsToggled för rack och noder kan applikationen nu exakt kontrollera när graferna ska visas eller döljas. Detta eliminerar behovet av att dubbelklicka på "visa graf" efter att användaren har stängt grafen, vilket tidigare var ett problem på grund av att grafkomponenten inte meddelades om det interna tillståndet.

En funktionalitet som var viktig att implementera i gränssnittet var ett verktyg för att snabbt kunna identifiera och undersöka noder av särskilt intresse. Därför infördes en funktion för att hantera varningar relaterade till noder och rack i systemet som syftade till att ge bättre överblick av kritiska händelser. Implementationen inkluderade en varningsmeny i gränssnittet, åtkomlig via en knapp, som användaren kan öppna och stänga efter behov. När varningsmenyn är öppen presenteras en organiserad lista över alla noder som har genererat varningar. Varje nod visas tillsammans med antalet associerade varningar, och användaren kan expandera varje nod för att se detaljerad information om varje enskild varning. I koden hanteras denna funktionalitet genom att hämta varningsdata från backend-servern via funktionen getWarnings(). Denna funktion anropar fetchWarnings(), som returnerar en lista över varningar.

4.4. Datahanteringen

En stor och viktig del av examensarbetet handlade om att se till att data kunde skickas mellan frontend och backend på ett så effektivt sätt som möjligt inom ramen av projektet. Tidigt i planering av examensarbetet fastslogs det i samband med diskussion med LUNARC att mängden data som hanterar inte är stor nog att det krävs någon databaslösning för att prototypen ska uppfylla de grundläggande kraven. Istället beslutades det att filer i CSV-format skulle kunna läsas in av backend-servern och omvandlas till JSON-format. Denna konvertering sker enbart en gång var 10:e minut vilket ger backend-servern och dess funktioner gott om tid att bearbeta data. Utöver detta var det möjligt att effektivisera processen ytterligare genom att kontrollera att CSV-filen innehåller nya data jämfört med tidigare, innan man börjar konvertera. Detta gjordes genom att generera ett hash-värde för CSV-data som jämfördes inför varje potentiell JSON-formattering.

Eftersom JSON-filerna skulle fungera som konfigurationsfiler och datamodell för frontend-gränsnittet var det viktigt att se till att enbart relevanta datamängder skickas från backend till frontend. Till en början hanterades en relativt liten mängd data, vilket innebar att all JSON-data kunde föras över från backend till frontend på en och samma gång. I och med att datamängden ökade fick tidigare datalösningar omarbetas så att mindre andelar data skickas åt gången. Det som behöver representeras med hjälp av grafer eller 3D-modeller hämtas och sparas temporärt i en map-struktur enbart om komponenten i fråga blir synlig, eller om uppdateringar begärs av användaren. Frontend-applikationen behövde på så sätt inte hämta mer data än vad som var hanterligt för webbläsaren. Mer konkret skapades tre asynkrona funktioner i frontend-applikationen för hämtning av partitionerade parameterdata. Dessa funktioner hämtade antingen alla parametervärden för alla noder i ett tidsögonblick, 24 timmars parametervärden för en nod eller de första och sista tidpunkterna i det aktuella 2-veckorsintervallet.

Ett viktigt krav på prototypens datahantering var att data från minst två veckor tillbaka skulle kunna hanteras. Eftersom LUNARC endast kunde tillhandahålla data när deras mätningar gav resultat, kunde inte ett komplett avsnitt med två veckors data utnyttjas från början, utan CSV-filen med alla mätningar fick växa till sig i två veckor innan datahanteringen kunde testas fullt ut. Från början fungerade en veckas dataöverföring mellan frontend och backend utan några märkbara fördröjningar. Det var dock när två veckor skulle hanteras som svarstiderna påverkades avsevärt. På grund av detta skapades en uppdaterad variant av CSV till JSON-konverteringen. I stället för att spara hela två veckors data i en stor JSON, partitionerades filen upp i två filer, en för varje vecka, eftersom det redan var känt att en veckas data kan hanteras snabbt av systemet. För att undvika att båda filerna söktes igenom vid hämtning av mindre andelar data så sparades den första och sista tidpunkten i vecka ett varje gång nya data hämtades in. Detta kunde sedan användas för att snabbt kontrollera vilken av filerna som skulle sökas igenom när frontend bad om att få data för ett visst datum skickat till sig.

5. Resultat

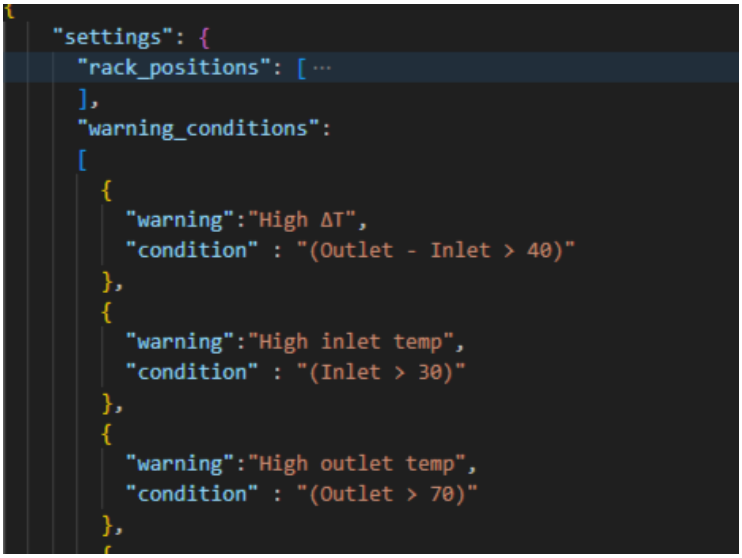
I detta kapitel beskrivs projektets resultat i relation till metod- och analyskapitlen för att illustrera vilka utfall som kan kopplas till särskilda beslut under examensarbetet gång. Eftersom examensarbetet handlade till stor del om 3D-användargränssnitt så kommer resultatet att presenteras med hjälp av skärmdumpar av applikationen i olika tillstånd. För övrigt är detta kapitel indelat i två delkapitel: ett för applikationens backend och ett för dess frontend. Figurerna i frontend-delkapitlet är manipulerade för att dölja information om nod- och rack-namn, samt positioner i rummet.

5.1. Backend

Vid examensarbetets början diskuterades backend-lösningar med handledarna hos LUNARC, där allt ifrån databaser, olika filformat och programmeringsspråk berördes. I slutändan beslutades det att backend skulle begränsas till en enklare lösning involverande CSV-filer som input från mätningar gjorda i den fysiska datorhallen och JSON-filer för förmedling av data till frontend-applikationen, d.v.s. ingen databaslösning. Detta beslut fattades på grund av 3D-gränssnittets många komponenter och höga grad av interaktivitet, samt det faktum att examensarbetarna hade väldigt begränsad erfarenhet av denna typ av programutveckling, vilket beräknades ta en större del av utvecklingstiden. Utöver detta skulle applikationen klara av presentationen av minst 2-veckors data i användargränssnittet, vilket uppfattades av handledarna som hanterbara datastorlekar för att skriva och läsa JSON-filer i backend. Detta visades sig vara en korrekt bedömning, eftersom den slutgiltiga datahanteringen kunde genomföras med hjälp av manipulering av endast JSON-filer.

Värt att nämna om JSON-filerna är att de tekniska lösningarna för uppdelning av data i mindre API-förfrågningar från frontend kanske inte är de absolut mest optimala. De lösningar som har valts har mer att göra med att särskilda prestandakrav skulle uppnås gällande hastigheten vid uppdatering av data. Den övre gränsen sattes till 3-5 sekunder, där manipulation av reglaget (vilket innebär ny inhämtning av ögonblicksdata eller en dags data) inte skulle trigga datauppdateringar som tar längre än 5 sekunder. I det värsta scenariot där användaren hämtar data för en tidpunkt för alla noder i ett rack samtidigt som nodens 2D-graf-fönster är öppet, vilket triggar hämtning av 24 timmars datapunkter för den noden, blir svarstiderna mellan 2000 och 2500 millisekunder på ett vanligt hemnätverk med bredbandshastighet på 100 Mbit/sekund.

Eftersom en databaslösning saknades för prototypen, gällde det att hitta effektiva sätt att partitionera data i JSON-filerna. Målet med datahantering mellan frontend och backend var att så få förfrågningar skulle ske, med så små datamängder som möjligt, utan att äventyra prototypens funktionalitet. Eftersom sökning genom en veckas JSON-data för att hitta en specifik tidpunkt kunde ske utan någon märkbar tidsfördröjning, till skillnad från att bearbeta hela två-veckorsintervallet, behölls lösningen där CSV-filen skrivs om till två JSON filer, i stället för en enda stor fil. Förutom detta sparades första och sista tid och datum i den första veckan för att snabbt kontrollera vilken fil som ska genomsökas av funktionerna och bibehålla samma tidskomplexitet för genomsökning av båda filerna.



Figur 5.1: Exempel på konfigurationsfilens struktur och innehåll. Av säkerhetsskäl kan ingen information om rackens namn och positioner visas i detta exempel.

Utöver uppdelning av data i lätthanterliga tidsintervall, så behövdes det även en bra struktur på JSON-data för att minska onödiga datahämtningar från frontend. I slutändan användes en kombination av JSON-konfigurationsfiler och JSON-datafiler innehållande parameterdata (se figur 5.1) . En av konfigurationsfilerna innehåller rackens namn, koordinater i 3D-rummet och booleska uttryck som triggar varningar i användargränssnittet. Denna fil ändras manuellt av administratörer för att manipulera rack och varningar ifall nya särskilda förhållanden behöver undersökas vid något tillfälle.

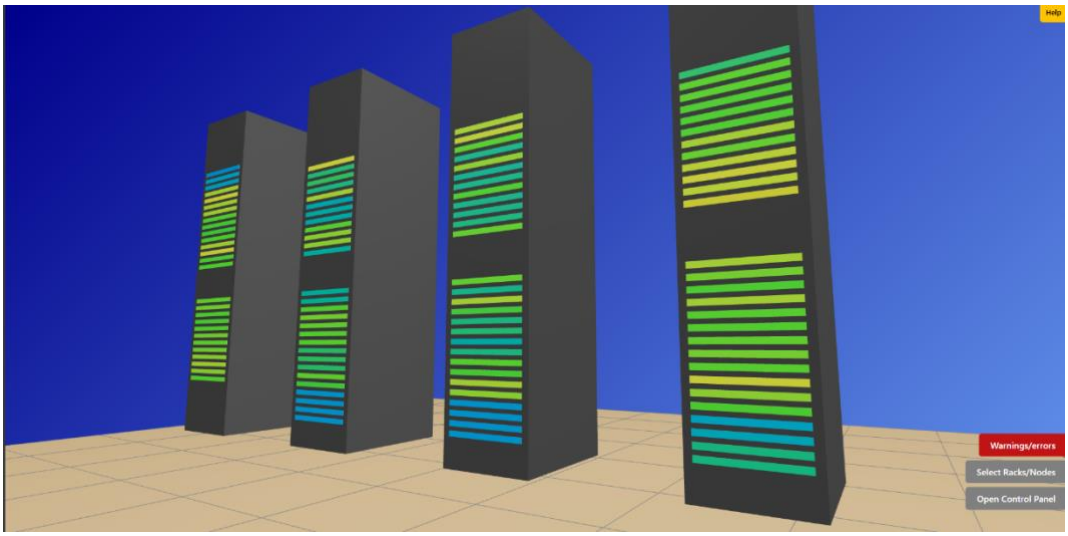
En annan fil innehållande noder, racktillhörighet och dess position i racket finns också, men detta baseras på inläsning av CSV-data för att tillåta automatisk förflyttning av noder i 3D vid förflyttning i den fysiska datahallen. Denna data behöver bara hämtas av frontend en gång var 10:e minut, eller vid uppstart av applikationen för vilket sparar in på nätverksprestanda.

Som tidigare beskrivet i analyskapitlet fanns det tre viktiga funktioner i frontend-applikationen som kommunicerade med endpoints i backend. Endpoints och deras ansvarsområden var följande:

1. En endpoint med nod-namn som in-parameter som hämtar alla parameteravläsningar för en dag. Denna metod anropades när en användare får upp graf-fönstret för en specifik nod, där trender över tid sedan kunde visualiseras. Denna funktion är relativt tidskrävande och bör således bara anropas vid selektering av nod.
2. En endpoint med datum som in-parameter (enl. Formatet “YYYY-MM-DD-HH:MM”) som returnerar alla noders parametervärden för en enda avläsning. Denna endpoint skapades för reglaget när en användare vill stega fram- och tillbaka i tiden och se hur temperaturvärden ritas om i 3D-gränssnittet.
3. En endpoint som hämtar de äldsta och senaste tidpunkterna för att reglagets tidsintervall ska kunna kalibreras ordentligt. Tanken med detta var att dessa två tidpunkter är de enda som behöver kännas till då varje mätning görs med 10 minuters mellanrum. Tidsåtgången för dessa två strängar är försumbart liten och sker en gång var 10:e minut.

5.2. Frontend

Frontend-delen av prototypen utgjorde den större delen av projektet eftersom mycket arbete krävdes för att se till att olika parametervärden kunde visualiseras på ett intuitivt sätt. Implementeringen av 3D-modeller var en viktig del av att undersöka om avläsning av särskild kontextuell information blir tolkningsbar (t.ex. temperaturskillnader i noder baserat på proximitet till andra noder). Det tredimensionella formatet gav dock även upphov för ytterligare interaktivitet som krävde mycket finjustering av tillståndsvariabler för att se till att markering av rack och noder kunde ske på ett naturligt sätt.

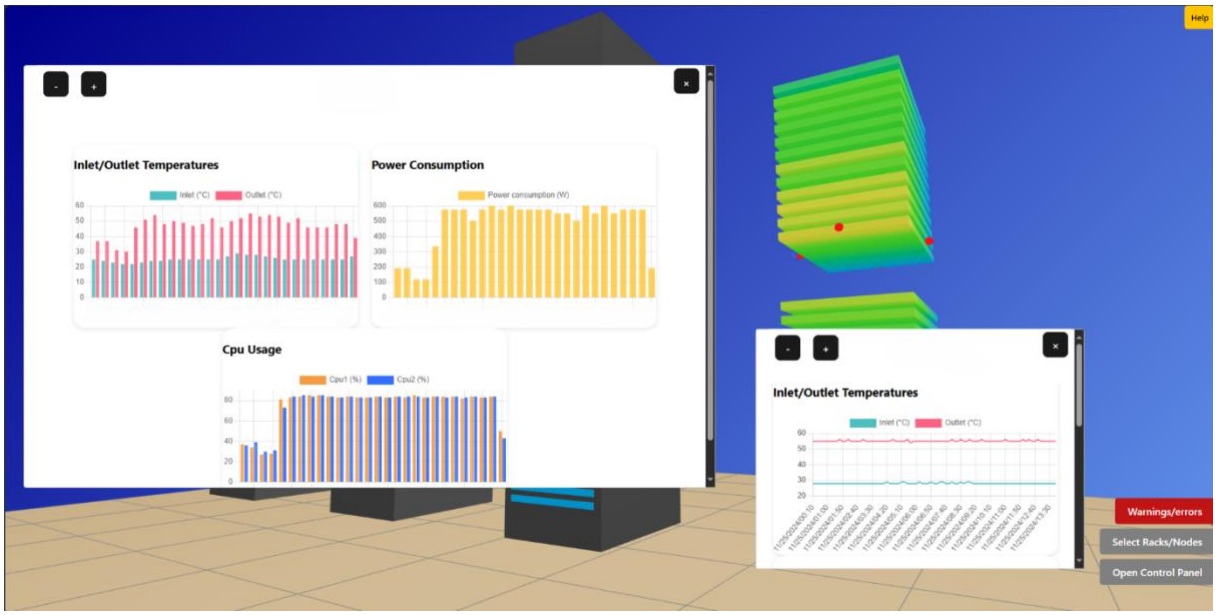


Figur 5.2: Här visas en manipulerad variant av 3D-gränssnittet.

Rack och noder består av enkel geometri för ökad prestanda och läsbarhet. Användaren möts av fyra meny-alternativ, vilket illustreras i figur 5.2: I nordöstra hörnet kan hjälpmenyn aktiveras, medan menyknappar i sydöstra hörnet triggar varningar och felmeddelanden, 2D-menyn för selektering av rack och noder samt kontrollpanelen som innehåller tidsreglaget.

Ett viktigt mål med användargränssnittet var att se till att användaren har möjlighet att själv välja hur mycket information som ska visas på skärmen, då all information inte alltid är relevant att ha synlig. En annan viktig aspekt gällande gränssnittets design var att se till att inte belasta användarens korttidsminne, genom att se till att man kan fritt kan växla mellan interaktion med 2D- och 3D-komponenter.

Rent tekniskt fungerar menyerna och graferna som bestående HUD-element som ger användaren möjlighet att navigera, avläsa och tolka 3D-modellens datarepresentation, samtidigt som information som enklast illustreras av grafer också finns synlig. Interaktion med 3D-modellen triggar även uppdateringar av 2D-element, vilket är tänkt att skapa upplevelsen av att 2D- och 3D är sömlöst kopplade till varandra.

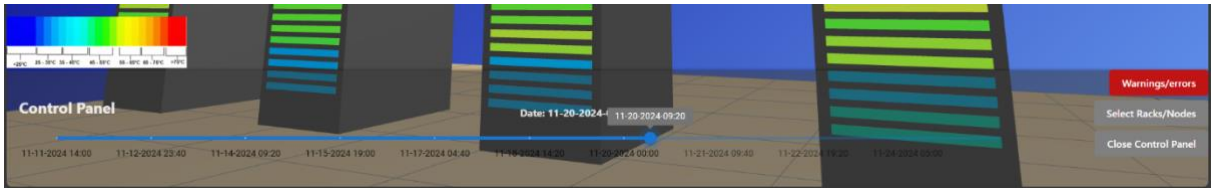


Figur 5.3: Markering av rack triggar graf-fönstret som visas till vänster, samtidigt som racket blir genomskinligt.

Graferna i rack-fönstret visar olika parametervärden för ett särskilt ögonblick i tiden för alla noder som tillhör racket. Grafen till höger visar en hel dags datapunkter för en markerad nod. Nodens graf-fönster innehåller samma parametertyper som i rackets fönster, men med linjediagram. Namn på rack och noder har även dolts p.g.a. säkerhetsskäl, vilket illustreras i figur 5.3.

Ett viktigt krav för prototypen var att användaren skulle kunna manipulera tiden med hjälp av 2D-gränssnittet vilket skulle uppdatera grafiska element med data från den valda tidpunkten. Det beslutades tillsammans med LUNARC att det mest rimliga alternativet hade varit ett reglage där varje steg representerade ett 10-minuters-intervall. Detta åstadkoms med hjälp av react-ramverket ”MUI”, som gav tillgång till en stilren reglage-komponent med väldokumenterade konfigurationsalternativ. Uppdatering av reglagets position uppdaterade ett värde som låg mellan den första och sista tidpunkten i 2-veckors-datan. Om tiden ändras, uppdaterades en variabel med det nya värdet från reglaget, varpå olika kontroller kunde utföras med s.k. ”useEffect” i react-biblioteket.

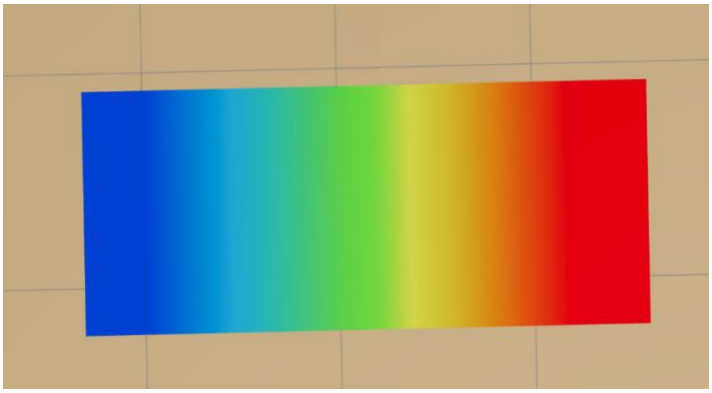
En ”useEffect” är en sorts callback-funktion som triggas om variabler som kopplas till funktionens interna vektor med beroenden ändras. Generellt kunde ”useEffect-”funktioner användas genom hela utvecklingen för att förhindra konflikter mellan komponenter och deras beteenden. I tidsreglagets fall så var reglagets uppdaterade tidsvariabel en del av den beroendevektor som triggade hämtning av data för en ny tidpunkt. Med hjälp av denna logik kunde man hålla hämtning av all ögonblicksdata för alla noder separat från hämtning av en nods data för en hel dag. Detta åstadkoms genom att se till att en dags data bara hämtades om en nods graf finns synlig och datumets dagsvärde ändrades, vilket begränsade onödig hämtning av api-förfrågningan till backend.



Figur 5.4: Tidsreglaget visar aktuell tidpunkt och uppdateras när användaren släpper markören.

Tidsreglaget har även grafiska element som informerar användaren om vilken tidpunkt som just nu är aktuell. Tid och datum uppdateras först när användaren släpper reglaget. Ovanför reglagets kontrollpanel visas en enkel mall för nodernas temperaturskala i syfte att informera användaren om temperatur och färgförhållanden i 3D-modellen, vilket illustreras i figur 5.4.

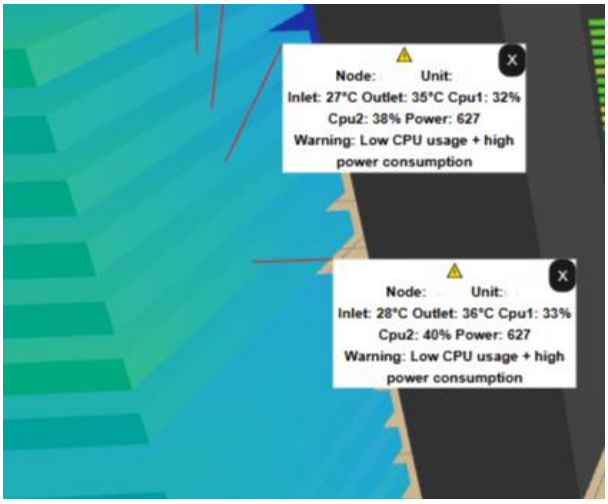
Nodernas temperatur-färgskala baseras på en enklare analys av den samtagna temperatur-mätdata från LUNARC, samt vad som gäller generellt för elektriska komponenter i datorer. Temperaturer under 25°C ger mörkblå färg och representerar särskilt låga temperaturer, medan rött indikerar att temperaturer överstiger 75°C, vilket innebär nivåer som potentiellt kan skada komponenter. Färgskalan beskrivs mer detaljerat i figur 5.5.



Figur 5.5: Färgskalan på en 3D-nod.

I figur 5.5 visas hela möjliga färgskalan på en 3D-nod. Skalan går från blå (<25°C), ljusblå (25–35°C), blågrön(35–45°C), grön (45–55°C), gul (55–65°C), orange (65–75°C) och slutligen röd (>75°C). Skalan skalas och anpassas automatiskt baserat på in- och ut-tempereaturernas parametervärden, vilket beskrivs i analyskapitlet.

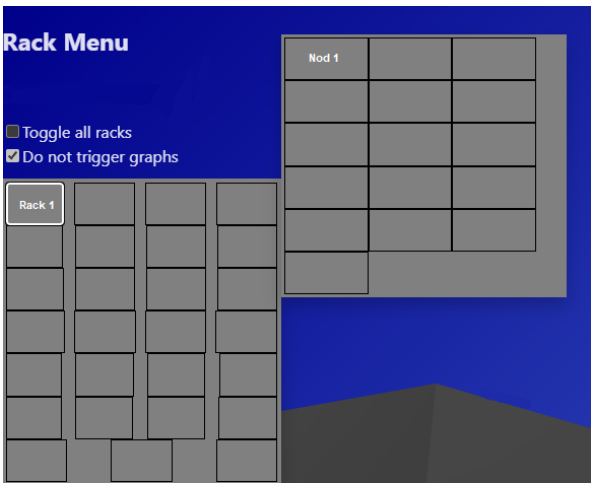
En annan central del av frontend-utvecklingen var att utforma ett enklare varningssystem till prototypen. När tidsreglaget uppdateras och nya data inhämtas och renderas, sker också en kontroll av olika parametervärden baserat på config-filens booleska uttryck. Om dessa villkor uppfylls triggas rendering av informationsrutor i 3D-rummet. Detta gjordes baserat från feedback från LUNARC efter en demo-granskning och var tänkt att ytterligare utnyttja 3D-gränssnittets särskilda spatiala egenskaper.



Figur 5.6: Varningar i 3D-gränssnittet.

I figur 5.6 visas ett exempel på hur varningar kan se ut i 3D-gränssnittet. Varningarna dyker upp i form av enkla HTML-komponenter vars koordinater kopplas till den tillhörande nodens koordinater. Detta gör det möjligt att dra röda linjer som markerar vilken varning som tillhör en nod.

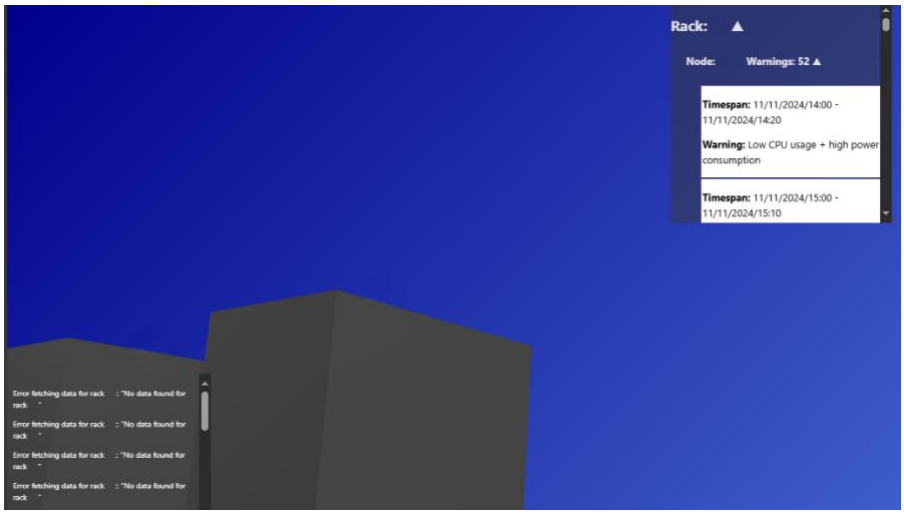
Rack-menyn skapades i syfte av att kringgå eventuella navigeringssvårigheter som användaren skulle kunna uppleva med 3D-gränssnittet. Detta resulterade i en meny som kan visas som en HUD-komponent, där användaren får en överblick av vilka rack som kan interageras med och vilka noder som hör till varje rack. Detta var tänkt att ge användaren direkt tillgång till grafer tillhörande rack och noder så att man slipper navigera genom 3D rummet varje gång man vill interagera med rack och noder.



Figur 5.7: Rackmenyn listar alla rack och möjliggör visning av grafer och noder samt inställningar för genomskinliga 3D-rack och direkt markering av noder genom 2D-menyn.

I rackmenyn syns alla rack uppradade samtidigt. När användaren klickar på ett rack i menyn öppnas rackets graf-fönster samtidigt som en liknande lista med tillhörande noder triggas i menyn. Förutom direkt markering av komponenter, finns även en funktion som gör alla rack genomskinliga samtidigt och en funktion som hindrar att grafer triggas vid navigering i 3D. Dessa funktioner och deras användning illustreras i figur 5.7.

Den sista huvudsakliga menykomponenten som skapades var varnings- och felmeddelandemenyn. Eftersom LUNARC efterfrågade ett sätt att kunna få överblick över varningar utöver de små fönster som triggades i 3D-gränssnittet så skapades en meny som visar förprocessade data från backend, samt inbyggda verktyg för att underlätta felsökning med hjälp av applikationen. Dessa varningars data hämtas till frontend i samband med varje 10-minuters uppdatering och visar varningar grupperade efter längre tidsintervall. Om en och samma varning gäller för en nod minst två gånger i rad så utökas varningens tidsintervall med 10 minuter, vilket är tänkt att ge användaren ett verktyg för att direkt kunna identifiera tidsintervall och komponenter som triggar särskilt många varningar under två-veckors-perioden. Utöver varningarna visas även de datarelaterade felmeddelanden som genereras av frontend-applikationens olika fetch-funktioner.



Figur 5.8: Varningar grupperade per rack och nod, samt felmeddelanden vid tom data från backend.

I nordöstra hörnet presenteras varningar i en hierarkisk struktur där varje rack och dess tillhörande noder grupperas tillsammans. För varje varning anges det tidsintervall då den kontinuerligt triggades under de senaste två veckorna, vilket ger användaren en tydlig överblick över återkommande problem. I sydvästra hörnet visas systemets felmeddelanden, vilka aktiveras när backend returnerar tom data för specifika rack. Genom denna presentation, som framgår i Figur 5.8 får användaren insikt i både aktuella och historiska avvikelser i serverhallen.

I sin helhet består användargränssnittet av en kombination av interaktiva 2D- och 3D-komponenter utformade enligt kravspecifikationer framtagna tillsammans med LUNARC-handledarna, resultat från demo-granskningar tillsammans med anställda hos LUNARC, samt användartester. I samband med ett avstämningsmöte som ägde rum vecka 50 meddelade handledarna hos LUNARC att de tyckte att prototypen uppfyllde de uppsatta kraven. Hur detta relaterar mer direkt till frågeformuleringen, etiska aspekter och möjliga framtida förbättringar diskuteras mer i detalj i slutsatskapitlet.

6. Slutsats

Detta kapitel sammanfattar det viktigaste i resultatet och genom en diskussion utifrån problemformulering i avsnitt 1.4 samt aspekter som sekretess, hederskodex och framtida utvecklingsmöjligheter.

Prototypen som har utvecklats inom detta examensarbete adresserar de centrala frågorna i problemformuleringen och har resulterat i ett visuellt övervakningsverktyg för LUNARC:s HPD-resurser. Genom att integrera 3D-och 2D-gränssnit har den föreslagna lösningen öppnat upp nya möjligheter för övervakning och identifiering av termiska hotspots och resursförbrukning.

1. Hur sker övervakningen av resurserna idag?

Övervakning av resurserna hos LUNARC sker idag främst med hjälp av loggdata som generas av ett verktyg på deras loggdataserver för att bearbeta IPMI-data från nodernas sensorer. Detta verktyg omvandlar insamlade data till CSV-format som kan användas för vidare analys. En stor del av temperatur-analysen sker manuellt genom att bearbeta CSV/filerna i Excel. Därmed finns det för närvarande ingen automatiserad interaktiv visuell representation av noderna och rackens status. Detta gör det svårare att upptäcka avvikande värden eller korrelera data med fysiska termiska tillstånd.

2. Vilka parametrar skall visualiseras?

LUNARC har uttryckt ett behov av att kunna visualisera flera viktiga parametrar som är centralt för att övervaka resurserna:

- In-och ut temperatur på noderna
- CPU-användning
- Strömförbrukning

3. Vilka verktyg finns det för att inhämta övervakade parametrar?

Det data som har använts i detta examensarbete är den data som har tillgängliggjorts av LUNARC. på grund av säkerhetsbegränsningar har påverkat mängden data som fick användas i prototypen, därför användes det delmängder av loggdata i examensarbetet.

4. Vilket programspråk skall användas för att implementera systemet.

Programspråk som användes var JavaScript för att implementera både frontend och backend. Detta val baserades på språkets portabilitet och kompabilitet med de bibliotek och ramverk som behövdes för att utveckla en webbaserad lösning. Backend-servern utvecklades med Node.js och ramverket Express.js, vilket erbjöd enkel hantering av API-anrop och konvertering av data från CSV till JSON-format. Frontend-applikationen utvecklades med React, vilket möjliggjorde en komponentbaserad och dynamisk användarupplevelse.

5. Vilka bibliotek och verktyg finns det för att implementera 3D visualiseringen?

JavaScript-biblioteket Three.js användes för att implementera 3D-visualiseringen av serverhallen. Biblioteket erbjuder verktyg för att skapa interaktiva 3D-modeller direkt i webbläsaren. Med hjälp av Three.js blev det enklare att skapa rack och noder samt hantering av texturer och gradienter. Ytterligare verktyg som användes inkluderade:

- Chart.js användes för att skapa grafer i 2D-gränssnittet
- MUI användes för att skapa stilrena komponenter såsom tidsreglaget
- PapaParse användes för att konvertera CSV-data till JSON-filer på backend

6.1. 3D-gränssnittet

Resultatet visar att prototypen uppfyller kravspecifikationen genom att ha funktioner som temperaturgradienter på 3D-gränssnittet, visualisering av parametrarna i 2D-gränsnittet, dynamisk hantering av varningar och felmeddelanden, samt dataöverföring mellan frontend och backend. Prototypen kan förhoppningsvis utöka LUNARC:s övervakningsmöjligeter gällande överhettning, vilket kan säkerställa mer effektiv drift. I sin tur kan detta potentiellt minska kostnader för energiförbrukning och underhåll samt minska risken för hårdvaruskador. Detta är särskilt betydelsefullt i en tid där serverhallens energianvändning är en alltmer central fråga ur ett miljöperspektiv.

Även om prototypen har granskats och godkänts av handledarna hos LUNARC, så identifierades givetvis nya områden som kan förbättras. Gradienterna och texturerna används för att visuellt representera temperaturförändringar på noder och det ger en snabb överblick över kritiska problem men systemets användarupplevelse kan fortfarande förfinas genom att öka större precision vid interaktion med 3D-komponenter.

6.2. 2D-gränssnittet

De varningsfönster och varningsvillkor som triggas kan hjälpa vid identifiering av problem med noder, men dessa hade kunnat göras mer intuitiva genom att tillföra fler visuella signaler eller mer detaljerad felsökningsinformation.

En sorteringsfunktion där användaren kan sortera varningar efter prioritet, kategori såsom temperatur eller energiförbuktning hade gjort varning- och error-fönstret mer informativt. Att utnyttja automatiska kamerarörelser för att direkt ”hoppa” till intressanta noder hade också bidragit med ytterligare användbarhet. Utöver dessa ändringar hade man kunnat göra gränssnittet ännu mer visuellt tydligt så hade man kunnat färga rack på olika sätt för att också representera situationer där särskilda data saknas.

Graferna i 2D-gränssnittet visar temperatur, energiförbrukning och nodens arbetsbelastning på ett brukbart sätt, men man hade kunnat lägga till fler parametrar såsom fuktnivåer, luftflöde och rummets temperatur. För att få ut så mycket som möjligt av 2D-gränssnitet hade en jämförelse-vy där möjligheten finns att visa flera noder och rack i samma graf underlättat jämförelse av deras parametrar.

6.3.Etisk reflektion

Under examensarbetet har etiska överväganden varit centrala för att säkerställa att arbetet bedrivits på ett ansvarsfullt och korrekt sätt. Arbetet har präglats av noggrannhet och ansvarstagande, där fokus legat på att uppnå balans mellan teknisk utveckling, konfidentiell datahantering och samhällsnytta. Särskild vikt har lagts vid sekretess och hederskodex för att säkerställa säker hantering av säkerhetsmässigt känsliga data och utveckling av en prototyp som uppfyller både LUNARC:s behov och bredare samhällsnytta. I det följande beskrivs mer utförligt de etiska aspekterna kopplade till sekretess och hederskodex som varit avgörande för projektets genomförande.

6.3.1. Sekretess

Under hela projektets gång har datainsamling och datahanteringen följt tydliga instruktioner från LUNARC. På grund av detta har riktlinjer angående sekretess inkorporerats som särskilda krav. En stor del sekretessproblemen har kunnat undgås genom att enbart behandla partiell data för ett mindre antal noder, inklusive noder som just nu är på väg att fasas ut ur systemet. Dessutom har projektet inte haft någon direkt tillgång till LUNARC:s loggdatabas, utan har enbart använt särskilt formaterade loggdatafiler, för att se till att projektet inte någonsin hanterar en komplett uppsättning data från systemet.

I alla delar av examensarbetet har det säkerställts att information som rör deras interna system, datamiljö och racken och servrarna placering etc. inte hamnar i orätta händer eller riskeras att exponeras för obehöriga. Alla loggfiler och ritningar som innehöll data för servrarna och rack klassades som konfidentiell information. Om det vid något tillfälle uppstod osäkerheter kring hur data kan hanteras, såg exjobbarna till att kontakta handledarna hos LUNARC för att få ett förtydligande.

Loggfilerna innehöll inga personuppgifter, men deras innehåll representerade känsliga data berörande LUNARC:s lokaler, rack- och nodpositioner som inte fick delas utanför arbetes ramar. Hanteringen av LUNRAC:s data aktualiserade även etiska övervägande kring säker lagring och integritet. Även om examensarbetet inte hanterade personuppgifter, är driftdata för serverhallar värdefull och kan potentiellt användas av obehöriga för att kartlägga sårbarheter i systemet. Prototypen är dock anpassat för framtida integration med loggdatabaser, där tillgång och hantering av data måste följa relevanta regelverk och standarder, som GDPR, för att skydda integriteten.

6.3.2. Hederskodex

Examensarbetet har utförts med förankring i hederskodexen med fokus på akademisk integritet och ansvarsfull utveckling. I ingenjörernas hederskodex konstateras följande: “Ingenjören bör i sin yrkesutövning känna ett personligt ansvar för att tekniken används på ett sätt som gagnar människa [...] och samhälle”. (Sveriges Ingenjörer, 2025)

Arbetet har genomförts med noggrannhet och akademisk integritet för att se till att balansera kundens önskemål mot prototyputvecklingens verklighet. Alla källor och insamlad information har redovisats tydligt, och rapporten utformats för att försöka sakligt och trovärdigt beskriva resultat utan att spekulera.

Examensarbetet har strävat efter att utforska nya teknikområden för att bidra till ökad samhällsnytta. Målet har varit att skapa lösningar som underlättar övervakningen av LUNARC:s serverhallar och stödjer potentiellt snabbare felsökning och effektivare användning av resurser, vilket kan gynna både organisationer och i förlängning den forskning som bedrivs med hjälp av beräkningsresurserna. Därför tillförs det ett värde som går bortom organisationens direkta behov och som kan inspirera andra initiativ inom området.

Under processens gång har examensarbetarna bedrivit ett tydligt ansvarstagande gentemot LUNARC. Regelbunden rapportering och hantering av feedback har säkerställt att arbetet följer de riktlinjer och krav som ställts av LUNARC. Datahanteringen har skett i enlighet med deras instruktioner och begränsningar för att skydda känslig information och prototypen har utvecklats med ambitionen att möta deras behov och förväntningar, trots de resursbegränsningar som har rått.

6.4.Framtida utvecklingsmöjligheter

Prototypen uppfyller kravspecifikationen och är fullt fungerande men den bör givetvis ses som ett underlag för vidareutveckling, snarare än en färdig kvalitetssäkrad produkt. Ett tydligt utvecklingsområde är förbättring av datahanteringen. För närvarande använder prototypen JSON-filer som ett praktiskt sätt att hantera mindre datamängder. Om systemet ska kunna hantera större mängder data, är det inte säkert att denna lösning kommer att räcka till.

Att implementera en relationsdatabas som PostgreSQL eller en NoSQL-lösning som MongoDB kan förbättra prestanda och underlätta integrering med ytterligare API:er. Dessutom kan det möjliggöra en mer avancerad dataanalys, vilket i sin tur kan underlätta identifiering av långsiktiga trender och samband mellan olika parametrar. Ytterligare en fördel med en databaslösning hade kunnat vara att det kan minska risken för redundans och fel i filhantering, samt att det kan öppna upp för automatisering av dataflöden från LUNARC:s sensorer.

Prototypen erbjuder utbyggnad för nya funktioner tack vare dess modulära design. Det ska med andra ord finnas utrymme att kunna anpassa systemet för nya behov såsom stöd för nya mätvärden, integrering av maskininlärningsmodeller för att förutse avvikelser.

7. Terminologi

2D	representation som använder två koordinataxlar (x, y) för att skapa bilder och visualiseringar utan djup.
3D	representation som använder tre koordinataxlar (x, y, z) för att simulera djup och rymd.
API	*Application Programming Interface*, uppsättning protokoll och verktyg för att bygga mjukvara och möjliggöra kommunikation mellan olika mjukvarukomponenter.
Backend	del av ett mjukvarusystem som hanterar databehandling, lagring och logik, oftast osynlig för slutanvändaren.
CSV	*Comma-Separated Values*, textbaserat filformat för att lagra tabulära data där värden separeras med kommatecken.
CORS	*Cross-Origin Resource Sharing*, säkerhetsfunktion i webbapplikationer som styr hur resurser delas mellan olika domäner.
Frontend	del av ett mjukvarusystem som är synligt och interagerar med slutanvändaren, inklusive grafiska gränssnitt.
IPMI	*Intelligent Platform Management Interface*, gränssnitt som används för att fjärrövervaka och styra hårdvara, till exempel serverkomponenter.
JSON	*JavaScript Object Notation*, lättviktsformat för att utbyta och lagra data som är lätt att läsa och skriva för både människor och maskiner.
MUI	*Material-UI*, bibliotek för React som tillhandahåller fördesignade användargränssnittskomponenter.

8. Källförteckning

1. Alkharabsheh, S., Fernandes, J., Gebrehiwot, B., Agonafer, D., Ghose, K., Ortega, A., Joshi, Y., & Sammakia, B. (2015). A brief overview of recent developments in thermal management in data centers. *Journal of Electronic Packaging, 137*(4), 1-16. https://doi-org.ludwig.lub.lu.se/10.1115/1.4031326
2. Axios (2024) *Axios – Promise based HTTP client for the browser and Node.js*. https://axios-http.com
3. Cabello, R. (2024). *Three.js documentation*. https://threejs.org/docs/
4. Chart.js (2024) *Chart.js – Simple yet flexible JavaScript charting library*. https://www.chartjs.org
5. Fulpagare, Y., & Bhargav, A. (2014). Advances in data center thermal management. *Renewable and Sustainable Energy Reviews, 43*, 982-990. https://doi.org/10.1016/j.rser.2014.11.056
6. Johnson, J. (2020). *Designing with the mind in mind: Simple guide to understanding user interface design guidelines* (2nd ed.). Elsevier.
7. Khalaj, A. H., & Halgamuge, S. K. (2017). A review on efficient thermal management of air- and liquid-cooled data centers: From chip to the cooling system. *Applied Energy, 205*, 1175-1178. https://doi.org/10.1016/j.apenergy.2017.08.037
8. Khronos Group. (2024). *WebGL specification* (Version 2.0). https://www.khronos.org/webgl/
9. Molina, J. P., González, P., Vanderdonckt, J., García, A. S., & Martínez, D. (2009). A space model for 3D user interface development. In V. Lopez Jaquero, F. Montero Simarro, J. P. Molina Masso, & J. Vanderdonckt (Eds.), *Computer-Aided Design of User Interfaces VI* (pp. 103-114). Springer London. https://doi.org/10.1007/978-1-84882-206-1_10
10. React (2024) *React – A JavaScript library for building user interfaces*. https://react.dev
11. Sterling, T., Anderson, M., & Brodowicz, M. (2017). *High performance computing: Modern systems and practices*. Morgan Kaufmann. https://doi.org/10.1016/C2016-0-02307-6

12. Zhang, K., Zhang, Y., Liu, J., & Niu, X. (2018). Recent advancements on thermal management and evaluation for data centers. *Applied Thermal Engineering*, *142*, 215-231. https://doi.org/10.1016/j.applthermaleng.2018.07.004
13. Sveriges Ingenjörer (2025), *Hederskodex* https://www.sverigesingenjorer.se/om-forbundet/organisation/hederskodex/

9. Appendix

I detta kapitel finns appendix med kravspecifikationen.

Appendix 1- Kravspecifikation

Terminologi

3D-modell	En digital representation av serverhallen och dess tillhörande enheter och komponenter.
CPU	Nodernas beräkningsenheter.
Serverhall	LUNARC:s serverhall.
Noder	Servrar tillhörande det sammanhängande HPC-prototypen.
Rack	De chassin som noderna monteras i.
Prototyp	Prototypen av det system som utgörs av alla frontend- och backend-komponenter.
Gränssnitt	De grafiska komponenter och funktioner som användaren möter.

Introduktion

Detta kravspecifikationsdokument är framtaget inom ramen för vårt examensarbete i samarbete med LUNARC. Projektets huvudsakliga mål är att utveckla en prototyp för visualisering och övervakning av de termiska förhållandena i LUNARCs HPD-serverhall. Genom att skapa ett gränssnitt med både 3D- och 2D- representationer av hallens komponenter ska prototypen göra det möjligt att monitorera termiska parametrar över tid, vilket kan bidra till att säkerställa effektiv drift i datacentret.

Prototypen ska dels visa en interaktiv 3D-modell av serverhallen, där rack, noder och kylsystem kan observeras, dels ett 2D-gränssnitt som presenterar loggdata och diagnostiska parametrar. Detta inkluderar temperaturmätningar, in- och utluftstemperaturer. Syftet med dessa funktioner är att underlätta identifiering av termiska hotspots och överhettning, vilket i sin tur kan förbättra kylningen och därmed minska driftkostnaderna.

Projektet genomförs enligt ett agilt arbetssätt med kravhantering baserad på

MoSCoW-prioriteringsmodellen. Denna prioriteringsskala kategoriserar kraven i fyra nivåer:

- **Must have (M)**: Absolut nödvändiga funktioner och krav som måste implementeras för att prototypen ska vara funktionell och uppfylla projektets mål.
- **Should have (S)**: Funktioner och krav som är viktiga men som kan utvecklas i en senare fas om tid eller resurser tillåter.
- **Could have (C)**: Funktioner som anses vara bra tillägg, men som inte är avgörande för prototypens grundläggande funktionalitet.
- **Won’t have (W)**: Funktioner och krav som inte kommer att inkluderas inom detta projekt men som kan övervägas i framtiden.

Denna prioriteringsskala säkerställer att de mest kritiska funktionerna implementeras först.

Dokumentet beskriver de funktionella och icke-funktionella kraven för både gränssnitten och backend-lösningen. Vidare finns krav på prestanda, skalbarhet och tillförlitlighet. Dokumentet kan komma att ändras i form av strykning eller tillägg av krav. Varje ändring kommer att gå igenom en valideringsprocess för att sedan dokumenteras i ett separat kravändringsdokument.

Sammanfattningsvis erbjuder detta dokument en översikt av de krav som styr utvecklingen av prototypen, med syfte att ge LUNARC bättre insyn i och kontroll över serverhallens termiska miljö och därmed bidra till en stabil och optimerad drift.

1. Funktionella krav

1.1.3D-gränssnitt

1.1.1. Prototypen ska bestå av en interaktiv 3D-modell över serverhallen. (M)

1.1.2. 3D-modellen ska uppdateras och renderas om dynamiskt baserat på indata.(M)

1.1.3. Rack, noder och kylsystem ska kunna positioneras i 3D-modellen på samma sätt som i den fysiska serverhallen. (M)

1.1.4. Det ska tydligt framgå vilket rack i hallen som varje nods 3D-modell tillhör. (M)

1.1.5. 3D-modellen ska uppfylla följande krav: (M)

- 3D-modellens vy ska tillåta rotering av modellen och in-och utzoomning både med tangentbord/mus och touch-kontroller.
- Serverhallens rack och noder ska representeras i 3D-modellen.
- Rackens fram- och bakdelar ska färgas enligt en färgskala från mörkblått till mörkrött för att indikera temperaturnivåer.
- Nodernas rack ska färgas enligt en färgskala från mörkblått till mörkrött för att indikera temperaturnivåer.
- Markering av en 3D-komponent ska trigga den tvådimensionella representationen.
- 3D-modellen ska renderas om i samband med användarens val av historisk ögonblicksbild.

1.2.2D-gränssnitt

1.2.1. Prototypen ska tillhandahålla ett tvådimensionellt gränssnitt för presentation av logg-data i form av följande diagnostiska parametrar. (M)

- Varje nods temperatur i C°
- In- och utluftens temperaturer i C°
- Energiförbrukning i watt (W)
- Arbetsbelastning
- Rutnäts-vyn ska tydligt partitioneras upp i överordnade nod- och rackgrupper.
- ~~Rutnäts-vyn ska färgkodas enligt en färgskala från mörkblått till mörkrött för att indikera temperaturnivåer.~~
- Det tvådimensionella gränssnittet ska innehålla grafer över diagnostiska parametrar.
- Graferna i det tvådimensionella gränssnittet uppdateras i samband med nytillkommen inhämtad logg-data.

1.2.2. 2D-gränsnittet ska innehålla följande kontroller:

(M) - Slå av/på genomskinliga rack-modeller.

- Markera individuellt rack.
- Markera individuell nod.
- Slider för rendering av historiska ögonblick i loggdatan.

2. Icke-funktionella krav

2.1. Prestanda

2.1.1. 3D-modellen ska kunna hantera realtidsuppdateringar av logg-data med max 3-5 sekunders fördröjning. (S)

2.1.2. 3D-modellen ska behålla svarshastigheter under 2 sekunder även vid stora datamängder. (S)

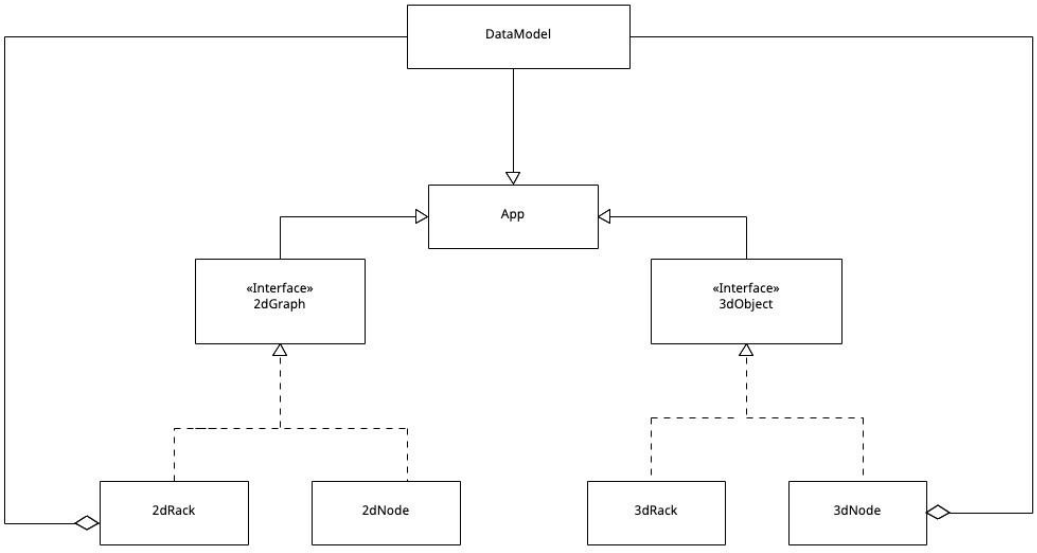
2.1.3. 3D-modellen ska renderas med lägst 30 bilder per sekund och högst 60 bilder per sekund. (S)

~~**2.1.4.** 3D-rendering ska vid behov optimeras med hjälp av dynamisk level-of-detail-hantering eller instance-rendering. (C)~~

2.2. Skalbarhet

2.2.1. Prototypens design ska tillåta horisontell skalbarhet för att stödja utökningar och ändringar bland noder och rack vid framtida utbyggnad av serverhallen.(M)

2.2.2. Prototypens komponenter kommer att struktureras enligt följande högnivå-klassdiagram: (M)



2.3. Tillförlitlighet

2.3.1. Prototypen ska vara åtkomlig via en webbläsare, anpassad för både större och mindre skärmar.(M)

2.3.2. Gränssnittet ska kunna stödjas av flera webbläsare, inklusive Chrome, Safari, Firefox och Edge. (M)

2.3.3. Backend ska kunna köras på operativsystemen Windows eller Linux. (M)

2.3.4. Prototypen ska ha felhantering som förhindrar att applikationen kraschar vid eventuella dataformat-fel.(M)

2.3.5. Felhanteringen ska generera läsbara meddelanden i 2D-gränssnittet där användaren kan tolka vad som har hänt. (M)

2.3.6. Vid mindre kritiska fel ska användaren ha alternativet att rendera om användargränssnittet. (S)

2.4.Användarvänlighet.

2.4.1. Prototypen ska ha ett intuitivt användargränssnitt som verifieras enligt användartester. (M)

2.4.2. Användargränssnittet ska innehålla informationsrutor med beskrivningar av interaktiva komponenter och funktioner.(M)

2.5. Säkerhet

2.5.1. All data som hanteras inom ramen för examensarbetet, inklusive bilder, loggfiler och annan information, ska behandlas som skyddsvärd information (M)

- Denna data får inte distribueras, delas eller kommuniceras med personer som inte är direkt involverade i examensarbetet.(M)
- All dataöverföring och lagring ska ske på ett säkert sätt för att förhindra obehörig åtkomst eller spridning (M)

3. Datakrav

3.1.Dataformat

3.1.1. Data som tillämpas av användargränssnittet ska formateras till JSON-filer. (S)

3.1.2. Loggdata-filen ska i JSON-format utgöra 3D-gränssnittets konfigurationsfil. (M)

3.1.3. Loggdata ska inkludera följande parametrar

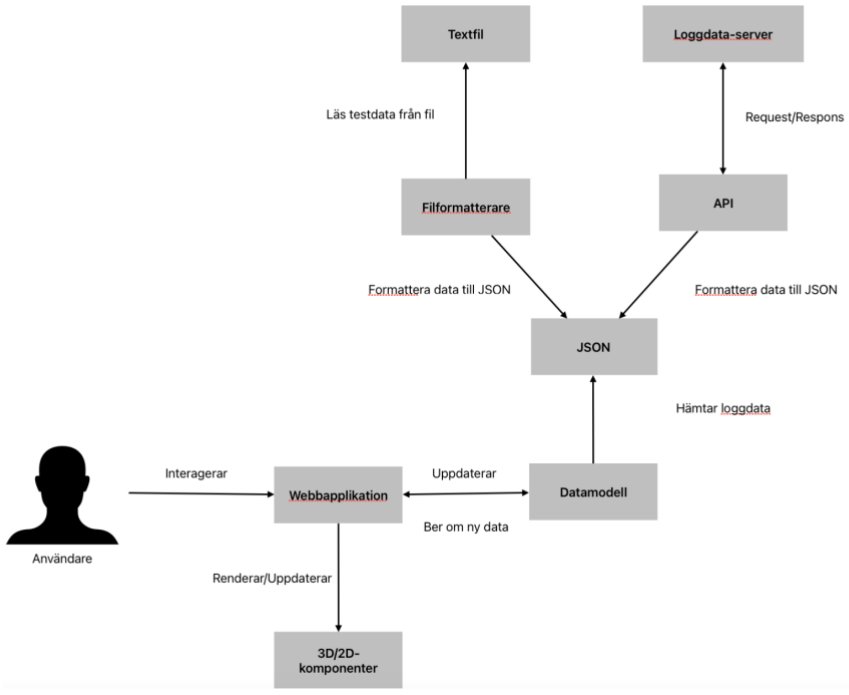
- CPU-parametrar (C)
- In-och utluftstemperaturer (M)
- Energiförbrukning (S)
- Arbetsbelastningen (S)

3.2.Datahantering

3.2.1. Prototypen ska kunna hantera och omvandla data från befintliga loggar till JSON-format. (M)

3.2.2. Prototypen ska kunna hantera historiska dataset upp till minst 14 dagar tillbaka i tiden. (M)

3.2.3. Dataflöden och interaktion ska kunna ske enligt diagrammet nedan:



3.3.Datafrekvens

3.3.1. Prototypen ska kunna hantera periodiska uppdateringar av data i tio-minuters-intervall.(M)

~~**3.3.2.** Prototypen ska kunna hantera realtidsuppdateringar av data. (C)~~

3.3.3. Användaren ska kunna välja att se historisk data för specifika tidsintervall. (S)

4. Systemkrav

4.1. Frontend

4.1.1. Frontend ska byggas med hjälp av HTML för strukturen, CSS för styling och JavaScript för logiken.(M)

4.1.2. React-ramverket ska användas för UI-uppdateringar och tillståndshantering. (M)

4.1.3. "Three.js"-biblioteket till JavaScript ska användas för rendering av 3D-modeller. (M)

4.1.4. "Chart.js"-biblioteket till JavaScript ska användas för att skapa interaktiva grafer och diagram. (M)

4.2. Backend

4.2.1. Backend-funktionalitet hos prototypen kommer att utvecklas med hjälp av "Express.js"-biblioteket via "Node.js"-plattformen för JavaScript. (S)

4.2.2. Av säkerhetsskäl ska backend-servern ta emot bearbetad simulerad data i lämpligt format utefter bestämmelser med LUNARC. (M)

4.2.3. Prototypens backend-design ska tillåta skalbarhet så att framtida API:er kan implementeras för direkt hantering av loggdata från interna servrar. (M)

5. Testning

5.1.Enhetstester

5.1.1. Enhetstester kommer att genereras periodiskt i takt med att nya funktioner utvecklas. (S)

5.1.2. Innan en ny funktion mergas med main branch i GitHub måste funktionen först genomgå ett regressionstest där alla tidigare enhetstester körs. (C)

5.2.Funktionstester

5.2.1. Funktionstester ska utföras informellt i samband med utvecklingen. (S)

5.2.2. Funktionstester ska utföras formellt med både utvecklare och intressenter på schemalagda tidpunkter.(S)

5.3.Användartester

5.3.1. Användartester ska utföras efter att särskilda milstolpar (se krav 7.1.1.) har uppnåtts. (M)

5.3.2. Användartester ska utföras formellt med intervjuer och checklistor (S).

5.4.Systemtester

5.4.1. Mer övergripande systemtester ska utföras formellt på schemalagda tidpunkter under projektets gång. (S)

5.4.2. Systemtester ska innehålla stresstester där större mängder data bearbetas i tidsintervall kortare än 10 min. (S)

5.4.3. Systemtester ska innehålla datatester där felaktig data genereras för att verifiera hantering av felaktiga dataformat(S).

6. Leveranskrav

6.1. Leveransdokument

6.1.1. Användardokumentation som förklarar hur prototypen används ska finnas i samband med leverans av prototypen. (S)

6.1.2. Teknisk dokumentation som beskriver prototypens ingående komponenter och funktioner. (M)

7. Projektkrav

7.1.Övergripande projektkrav

7.1.1. Projektet kommer att pågå i 15 veckor under veckorna 36 till 51.

7.1.2. Prototyputvecklingen kommer att bestå av följande faser: (M)

1. Design av intern datamodell och grundläggande 3D-modell.
2. Integrering av modell (färgschema osv.), dynamiskt uppdaterade 3D-komponenter och datahantering (MVC-struktur).
3. Färdigställning av 2D- och 3D-gränssnitt.
4. Systemtester, korrigeringar, optimeringar och leverans.

7.1.3. Projektet kommer att utföras agilt enligt krav 7.1.3. (S)

7.1.4. Projektets utvecklingsfas ska följa en variant av scrum-modellen där följande cykel repeteras veckovis:

- Product Backlog: Hela gruppen samlas på förutbestämd tid och planerar ännu ej genomförda uppgifter.
- Sprint planning: Hela gruppen diskuterar möjlig tidsåtgång och arbetsbörda för varje uppgift i en s.k. “product backlog”.
- Sprint: Utveckling av dokument, kod och testning.
- Daily Standup: Varje dag möts gruppen och ger en kort lägesrapport där ens pågående arbete presenteras.
- Sprint Retrospective: Efter varje sprint hålls ett möte där gruppen utvärderar den gångna veckan och lyfter fram olika framgångar och motgångar.

7.1.5. Fördelning av arbetsuppgifter kommer att genomföras med hjälp av “issues” i Github. (S)

7.2.Kravhantering

7.2.1. Kravhantering ska genomgå följande delsteg: (S)

- Elicitering: Information samlas in med intervjuer, prototyputveckling, efterforskning och brainstormingaktiviteter.
- Kravanalys: Insamlade krav analyseras och prioriteras enligt MoSCoW-metoden på regelbundna möten genom projektets gång.
- Kravspecifikation: Krav förs in enhetligt i kravspecifikationen baserat på kravanalysens resultat.
- Kravgranskning: I samband med nytillkomna funktioner och komponenter i prototypen kommer gamla krav att granskas för att verifiera deras giltighet.
- Kravändringar: Utefter förändrade omständigheter under arbetets gång kommer kravspecifikationen att uppdateras. Krav som inte längre anses vara giltiga, eller som formuleras kommer att strykas över och stå kvar i dokumentet. Ändringar kommer även att dokumenteras i ett kravändringsdokument.
- Validering och verifiering: Innan utveckling av prototypen påbörjas måste alla krav genomgå en valideringsprocess där för att säkerställa att de uppfyller projektets mål. Verifiering sker för att se till att prototypen fungerar korrekt i enlighet med tekniska krav.

7.3.Koddokumentation

7.3.1. Till varje klass eller komponent i prototypen ska det finnas kommentarer som presenterar klassens ansvarsområde och generella attribut. (S)

7.3.2. Metoder ska beskrivs med hjälp av kommentarer vid behov. Förklaringar kan delas upp i flera steg och i den ordning de sker i algoritmen.(S)

7.3.3. Koden ska granskas regelbundet inom gruppen för att säkerställa läsbarhet. Dessa granskningar kan ske informellt under antingen sprint-planning eller sprint-retrospective. (S)

7.3.4. Om möjligt ska länkar bifogas till resurser som kan hjälpa övriga utvecklare att förstå och använda koden.(C)

7.3.5. Versionshantering sker m.h.a funktioner i Github.(M)