Zimone Björnler
0107117301

Uppgiftsrapport

A virtual representation of the Art Institute of Chicago

I uppgiften har en webbapplikation framställts med hjälp av Electron för att kunna efterlikna och fungera likt en skrivbordsapplikation, och koden är skriven med objektorienterad JavaScript (ECMAScript 5). Applikationens syfte är att skapa en virtuell representation av museet Art Institute of Chicago för att göra det möjligt att uppleva den konst som finns där och att få en museiupplevelse utan att behöva besöka museet.

Applikationens syfte och vision är utformad efter museets API (Art Institute of Chicago 2023) och den information som det erbjuder men är skapat för att om möjligt lägga till fler källor till konst eller koppla till andra museer. Det fastställdes dock att för tjänstens tillverkning skulle det vara Art Institute of Chicago som låg i fokus och att skapa en virtuell miljö som efterliknar det som kan förväntas av ett verkligt museibesök.

Visionen för produkten är därför att skapa en förhållandevis realistisk och verklighetstrogen upplevelse digital museiupplevelse med hjälp av Art Institute of Chicagos API. Tanken för programmet är att det ska vara uppbyggt av klasser som ska vara så oberoende och självgående som möjligt för att optimera kodens återanvändbarhet (Weisfeld 2009, s. 91-92).

Genomförande

API

Informationen och bilderna som visas upp på väggen är hämtade från Art Institute of Chicagos egna API som innehåller deras egna uppsättning av konst men även lånade och tillfälliga konstverk. Innan utvecklingen av produkten började var det en process av att ta reda på hur API:et fungerade, vad det erbjuder i form av information, begränsningar, anrop m.m. och att sedan fastställa vad som ska användas för uppgiften och hur.

API:et tillhandahåller en stor mängd information om många konstverk och konstnärer, och över 300.000 individuella verk vilket gör det möjligt att använda det för olika projekt och ändamål. I detta projekt har det valts att framförallt använda det för att se konstverken och få information om dem likt som det visas på ett museum.

Arbetsprocess

Eftersom att projektet består av ett flertal delar krävdes det att en plan lades upp för vilka klasser som skulle finnas, hur dem skulle förhålla sig till varandra samt vad de skulle innehålla och hur de skulle fungera. Därför skapades vid arbetets start ett UML-diagram och en översiktlig vision av hur applikationen skulle se ut, fungera och utformas och klasserna tilldelades därför metoder och egenskaper som från en början ansågs passande deras syfte.

I det preliminära UML-diagrammet (se bilaga 1) delades programmet upp i åtta klasser som hade i uppgift att hantera olika delar och tillsammans bilda den fullständiga produkten. Under arbetet delades en del av klasserna upp i nya alternativt togs bort om dess tänkta uppgift bättre passade att utföras av en annan klass.

Eftersom att ett av produktens mål var att efterlikna ett verkligt museum delades programmets huvuddelar slutligen upp i tre klasser: Gallery, Lobby, Room och ArtWall. Dessa utgör tillsammans kärnan av tjänsten som sedan består av och använder sig av andra klasser och objekt för att till slut skapa den slutgiltiga produkten.

Det krävdes mycket beslutstagande kring hur saker skulle se ut och fungera; det handlade om att skapa god interaktivitet, att göra det realistiskt och samtidigt kontinuerlig genom hela applikationen. Den slutgiltiga produkten använder sig i större delen av fallen av pilar (som knappar) för att navigera innehållet men när det gäller visa innehållet (konstverk och informationstavlor) är elementen klickbara. Detta är både för att det ska se och upplevas mer verklighetstroget, för att inte skärmen ska uppfattas som plottrig med massa pilar och för att det enkelt skulle gå att visa ett specifikt individuellt element. Det är nu möjligt att visa ett specifikt konstverk eller informationstavla genom att endast klicka på det istället för att det skulle finnas en pil till varje element eller att behöva klicka för att ta sig fram till väggen och sedan navigera sig till rätt element.

För att förbättra användarupplevelsen var det viktigt att vara kontinuerlig mellan sidor och att ge användaren feedback i det den gör och vad som händer (Wong 2023). Detta görs bland annat genom att fokusera på väl implementerade mikrointeraktioner genom att dela upp interaktioner i mindre delar eller att lägga till interaktioner som utför specifika saker i syfte att förbättra upplevelsen (Saffer 2013, s.5). Det har exempelvis gjorts kopplat till sökrutan, där har knappen som används för att söka ett syfte och ger användaren feedback när något har sökts genom att ändra utseende och ändrar sedan tillbaka till en sök-ikon när användaren klickar i sökrutan på nytt.

I efterhand har det framkommit alternativa lösningar när det exempelvis gäller navigering visning av konstverk som möjligtvis hade kunnat göra applikationen mer lätthanterlig, effektiv och verklighetstrogen. Det hade exempelvis varit möjligt att hantera visningen av konstverk med endast musklick; användaren kunde isåfall vänsterklicka för att zooma in på ett element (konstverk eller dess informationstavla) och sedan högerklicka för att zooma ut bilden och se hela rummet. Detta hade medfört en mindre hantering av pilar och mer kontinuerligt men kunde skapa problem om användaren använder sig av en dator med musplatta eller annan touch och skulle kräva någon slags information för att användaren ska förstå hur produkten ska navigeras.

Försök gjordes för att skapa en övergång mellan rummen som skulle liknas vid att gå från ett rum till ett annat men resultaten var inte framgångsrika och därför lades det åt sidan för att istället fokusera på andra delar. Vid vidare utveckling skulle detta vara ett stort fokus då det även skulle förbättra upplevelsen om bilderna kunde laddas in medans vyn byts och på så sätt eliminera väntetiden. Detta problem gjordes det även försök till förbättring genom att visa

ikoner som visar att bilderna håller på att laddas in men detta resulterade bara i att användaren tyckte att det blev ryckigt eftersom att inladdningen oftast var ganska kort.

Produkt

Gallery representerar hela det virtuella museet och är därför också programmets basklass och initieras automatiskt när sidan laddas in och är statisk vilket gör den tillgänglig under hela exekveringstiden och från alla delar av programmet (Stefanov & Sharma 2013, s. 112-113). Klassen sköter den allmänna logiken för programmet såsom att skapa, visa och byta rum; det vill säga att navigera och kontrollera applikationens delar. Eftersom att Gallery hanterar navigeringen och visningen av rummen är det också där som rummen och lobbyn tilldelas navigeringen i form av pilar och händelselyssnare.

Vid programmets start skapar Gallery Lobby:n och visar den som startsida, och hanterar sedan metoderna för att skapa och ta bort rum, visa nästa och gå tillbaka till ett tidigare rum samt att visa och dölja Lobby:n. För att göra detta sparas användarens aktuella position i förhållande till rummen och det senaste aktiva rummet vilket gör det möjligt för användaren att visa Lobby:n och sedan gå tillbaka till det rum som senast besöktes.

Klassen Lobby är också en statisk klass och representerar museets lobby; det rum som användarna först anländer till när de besöker det virtuella museet varifrån dem sedan kan gå vidare in i museets rum. I lobbyn finns en informationstavla där information om galleriet går att hitta, samt en sökruta för att kunna hitta specifika verk och konstnärer eller för att söka efter någon genre, ord, tema eller plats. Sökfunktionen använder sig av API:ets egna sökfunktion för att hämta data utifrån det användaren skriver i sökrutan.

Room är en klass som representerar ett rum i museet och skapades egentligen för att även vara föräldrar-klass till Lobby men eftersom att Lobby-klassen, och i och med det syfte som den skulle uppfylla, skapades som en statisk klass kan den inte ärva från någon annan klass (Mozilla 2023). Därför är klassen inte så generellt byggd som den kunde ha varit men det finns inga svårigheter om klassen ska anpassas för att kunna användas till fler ändamål.

För att visa upp konstverken i Room skapas en instans av klassen ArtWall som anropar API:et och visar upp resultatet på rummets vägg. Antalet konstverk som visas anpassas efter fönstrets storlek och varje bild får en ram (av klassen Frame) som anpassas efter dess storlek, samt en informationstavla (av klassen ArtInfo) innehållande fakta om verket.

ArtWall hanterar som sagt anropet för att hämta datan och skapar individuella objekt av typen Artwork för varje hämtat verk och “hänger upp” dem på väggen i rummet. Om länken till verket (bilden) inte hittas anges en egen bild-fil som säger att bilden inte kunde visas, detta är ett problem som skulle behöva hanteras mer (läs mer under rubriken Brister). Klassen hanterar även visningen av konstverken och logiken bakom hur och när navigeringspilar och informationsrutor visas.

Konstverken (Artwork) innehåller sedan både bilden av verket samt dess tillhörande informationstavla (ArtInfo) och lägger även till en ram samt en informationstext som visas

när musen förs över elementet. Eftersom att ramen ska anpassas efter bildens storlek och form så kräver det att programmet behöver vänta på att bilden laddats in helt innan ramen lades till vilket löstes genom att skapa en metod som kollar om bilden är uppladdad och om inte så kallar den på sig själv igen.

Brister

Vid vidareutveckling av applikationen är det tänkt att fortsätta arbetet och vidga applikationens användningsområde samt optimera och förbättra den kod som redan är skriven. Bland annat är det planerat att tillföra ytterligare sätt att navigera och söka eller sortera bland galleriets innehåll och att vid sökning kunna visa hur många träffar/antal resultat en sökning gett.

När API:et är under uppdatering är inte länkarna till verkens bilder uppdaterade vilket resulterar i att en stor del av verken inte kan visas. Detta är ett problem som inte har hanterats på grund av att problemet var för komplext för att under arbetet lyckas lösa.

Källhänvisning

Art Institute of Chicago (2023). *Documentation* https://api.artic.edu/docs/#introduction [2023-03-16]

Mozilla (2023). *static* https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/static [2023-03-15]

Saffer, D. (2013). *Microinteractions - designing with details* Sebastopol, USA: O'Reilly Media, Inc.

Stefanov, S., Sharma, K., C. (2013). *Object-Oriented JavaScript* 2 uppl., Birmingham, United Kingdom: Packt Publishing Ltd.

Weisfeld, M. (2009). *The Object-Oriented Thought Process* 3 uppl., Berkeley, California, United States of America: Pearson Education

Wong, E. (2023). *User Interface Design Guidelines: 10 Rules of Thumb* https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb [2023-03-17]

Bilagor

Bilaga 1. *Preliminär UML (felaktiga relationer)*