IP3 – Webbutveckling; klientsidan (IB908C)

Peter Borgstedt ([pebo6883@student.su.se](mailto:pebo6883@student.su.se)), 2020

# Innehållsförteckning

[Innehållsförteckning 1](#_Toc49162941)

[Obligatoriska uppgifter utförda (8): 2](#_Toc49162942)

[2.3 Marginaler kantlinjer och paddning obligatorisk (CSS: 7) 2](#_Toc49162943)

[Frivilliga uppgifter utförda (8): 2](#_Toc49162944)

[Gesällprov 3](#_Toc49162945)

[Övergripande beskrivning av implementation 3](#_Toc49162946)

[Teknisk sammanfattning 3](#_Toc49162947)

[Tillämpning 3](#_Toc49162948)

[Övergripande beskrivning av applikationens funktionalitet 4](#_Toc49162949)

[Startsida 4](#_Toc49162950)

[Inloggning 5](#_Toc49162951)

[Registrering 6](#_Toc49162952)

[Huvudfönster 7](#_Toc49162953)

[Användarprofil 8](#_Toc49162954)

[Användarprofil 11](#_Toc49162955)

[Skapande av en kanal 13](#_Toc49162956)

[Bläddring av kanaler 14](#_Toc49162957)

[Kanaler 15](#_Toc49162958)

[Kort beskrivning över några av tekniker som används utanför kursens innehåll 18](#_Toc49162959)

[Webkomponenter 18](#_Toc49162960)

[REST 18](#_Toc49162961)

[Websocket 18](#_Toc49162962)

[Webtoken 18](#_Toc49162963)

[CSS Variabler 18](#_Toc49162964)

[Summering 19](#_Toc49162965)

# Obligatoriska uppgifter utförda (8):

1.1 Grunderna i HTML5 - obligatorisk (HTML5: 1)

1.2 Semantiska element för struktur obligatorisk (HTML5: 2)

1.3 Semantiska element för text obligatorisk (HTML5: 3)

1.4 Formulär obligatorisk (HTML5: 4)

1.6.2 Progressivt nerladdad media obligatorisk (HTML5: 5)

2.1 Grunderna i CSS 3 obligatorisk (CSS: 1-5, HTML: 6)

2.2 Text och listor obligatorisk (CSS: 6)

2.3 Marginaler kantlinjer och paddning obligatorisk (CSS: 7)

2.4 Bilder och bakgrunder obligatorisk (CSS: 8)

2.5 Navigering obligatorisk (CSS: 9)

2.6 Tabeller och formulär obligatorisk (CSS: 11)

2.7 Sidlayout obligatorisk (CSS: 12-14, 16-17)

2.8 Anpassning till olika enheter obligatorisk (CSS: 15, HTML 7)

3.1 Grunderna i JavaScript obligatorisk (JavaScript: 1-3, 16)

3.2 DOM-objekt och jQuery obligatorisk (JavaScript: 5)

3.3 Webbläsar-objekt obligatorisk (JavaScript: 4)

3.4.1 Effekter och animationer obligatorisk (JavaScript: 6)

3.4.2 Bildhantering obligatorisk (JavaScript: 7, HTML5: 8-9)

3.4.3 Navigering obligatorisk (JavaScript: 7)

3.4.4 Utvidgbara gränssnitt obligatorisk (JavaScript: 9)

3.4.5 Formulärhantering obligatorisk (JavaScript: 8, 10)

# Frivilliga uppgifter utförda (8):

2.9 Nerladdad font frivillig (CSS: 6, HTML: 6)

2.10 Övergångar frivillig (CSS: 10)

3.5.1 Webblagring frivillig (HTML5: 10)

3.5.2 Filläsning frivillig (HTML5: 10)

3.5.3 Databaslagring frivillig (HTML5: 10)

3.7.2 Bakgrundsprocesser frivillig (HTML5: 13)

3.7.3 Historiehantering frivillig (HTML5: 13)

3.7.6 Offline applikationer frivillig (HTML5: 11)

Eftersom ”Application Cache” är deprecated så löstes uppgiften med ”Cache API” och ”Service Workers” istället vilket är rekommendationen idag.

# Gesällprov

En chat-klient med inspiration från sådana klienter som Teams, Slack med flera.

Som gesällprov har jag valt att bygga en chat-klient (**front-end**) med inspiration från klienter så som Teams, Slack med flera. Gesällprovet har kombinerats med kursen **IP1** (internetprogrammering, stationära enheter) som kommer beröra implementationen för webservicen (**backend-end**).

## Övergripande beskrivning av implementation

### Teknisk sammanfattning

Webapplikationen är designad som en **single page application** (SPA) och är skriven i **JavaScript** (utan tredjeparts-ramverk) med stöd av **webkomponenter**.

Valet till att använda webkomponenter kändes naturligt i samband med kursen. Jag har tidigare velat testa det och utvärdera huruvida det kan vara något som kan ersätta ramverk som **Angular**, **React**, **Vuejs** vilka även de är komponentbaserade (dock med virtuella element). En viktig skillnad är att en webkomponent är ett riktigt element vilket i sin tur har fördelen med av att vara snabbare eftersom den inte har en massa ”socker” runt om sig.

Applikationen använder sig av ett flertal tekniker, därav många från kursen men även tekniker utanför; allt från CSS med variabler, animeringar, transitioner, avancerade selektorer, stylningar av flera slag till manipulering av DOM och Shadow DOM. Kommunikation och dataöverföringar sker med websockets och REST (via HTTP). Det sker konvertering av data mellan olika format. Bilder läses in, drag & drop, bilder strömmas tillsammans med text och mycket mer.

Websockets och REST andvänds för att kommunicera med servern, för detaljerad information hur detta fungerar och arkitekturen på baksidan se dokumentationen för webservicen.

### Tillämpning

Applikationen är uppdelad av vyer. En vy är en presentation av ett innehåll och varje vy består av en webkomponent, denna kan i sin tur vara sammansatt av flera andra webkomponenter med delad logik.

I och med att en webkomponent är ett HTML-element så har den naturligt stöd för händelser (events), utöver det har även en skräddarsydd event-hanterare (eventbus) implementerats för de händelser som behöver skickas till funktionalitet utom komponentens räckvidd; så som andra vyer som inte är synliga; tjänster som kommunicerar med servern med mera.

Varje webkomponent består av HTML och CSS, dessa är infogade ”**inline**” i koden (detta är enligt design).

All kod är skriven med **JavaScript ES6** och majoriteten av alla filer är strukturerade enligt **ES6 moduler**, d.v.s. funktioner kan exporteras och importeras, vilket möjliggör modularisering och återanvändning av kod via import.

Implementationen för tjänster som REST-anrop och transaktioner via Websocket är utformad i **ES6 moduler** vilket gör det enkelt att använda. Dessa nyttjar i sin tur **Websocket API** och **Fetch API**, vilka båda är en del av **HTML5**.

Det finns ett tredjeparts bibliotek som används; **CropperJs** vilket är en färdig JavaScript-lösning för att klippa bilder, denna används vid redigering av profilbilder.

All kod är väl beskriven med **javadoc** och referenser till källor där jag bland annat fått beskrivningar för hur vissa bitar bör implementeras (best practices med mera).

Det finns en del **TODO’s** som jag lämnat kvar medvetet för dokumentationens skull och hu r vissa saker kan eventuellt göras bättre.

## Övergripande beskrivning av applikationens funktionalitet

Som ovan beskrivet så är webapplikation byggd av vyer som alla har olika typer av funktionalitet, dessa kommer att kortfattat beskrivas nedan.

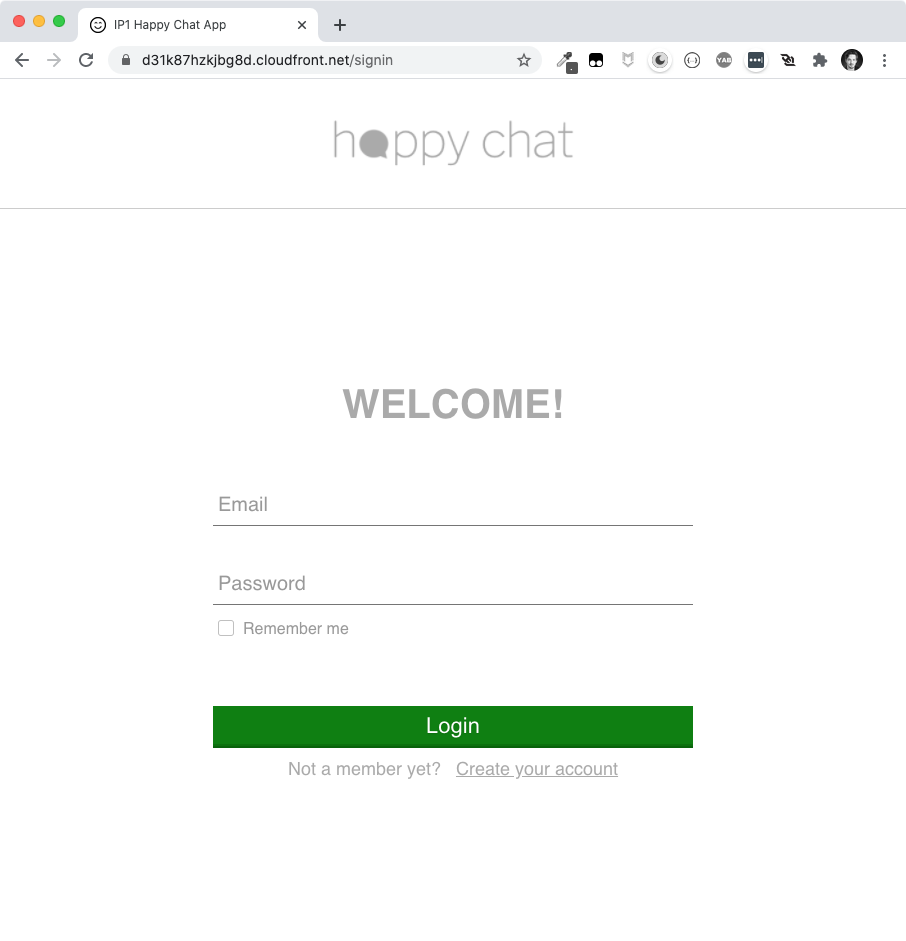
### Startsida

Startsidan består av en ”routing” som granskar huruvida användaren är inloggad eller inte.

1. Om användaren är inloggad (d.v.s. har en giltig token) så hänvisas denna till huvudvyn.
2. Om användaren inte är inloggad så hänvisas denna till inloggningsfönstret, som i sin tur även innehåller en länk för registrering.

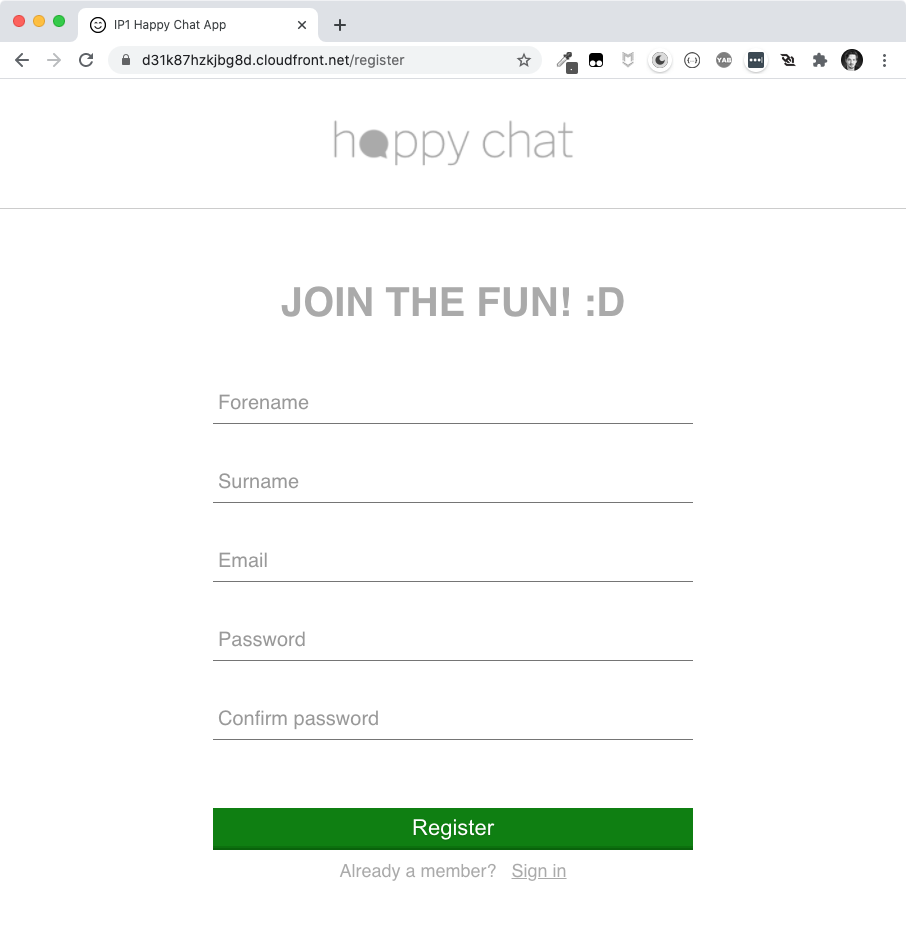
### Inloggning

Vyn består av ett formulär där användaren fyller i sina inloggningsuppgifter, därefter utförs en validering, om något inte skulle stämma så informeras användaren om detta, om uppgifterna stämmer så skickas detaljerna vidare via ett REST-anrop till servern, om allt går bra så returneras en webtoken. Ifall användaren inte har ett konto så finns möjligheten att klicka på en länk för registrering.



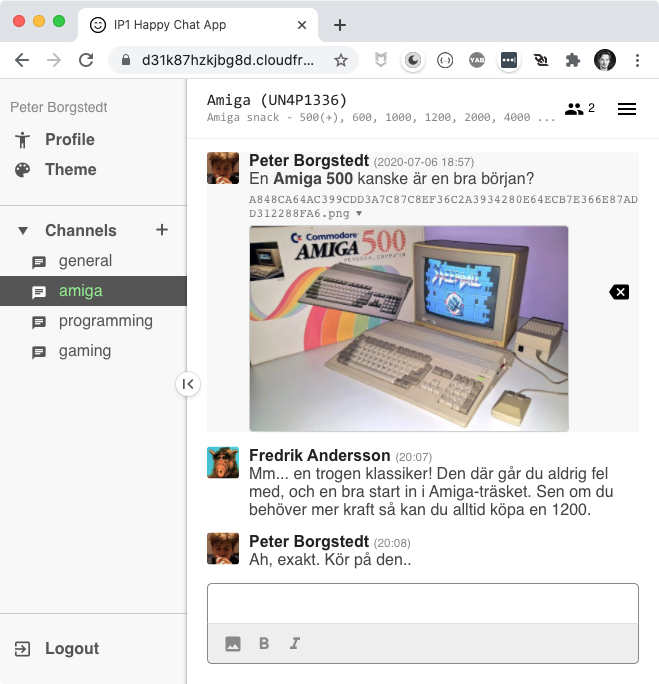
### Registrering

Vyn är näst intill identisk till inloggningen, dock med några ytterligare fält. Användaren fyller i sina uppgifter och därefter utförs en validering, om något inte skulle stämma så informeras användaren om det, om allt stämmer så skickas detaljerna vidare via ett REST-anrop till servern och om allt går bra så hänvisas klienten vidare till inloggningsfönstret.



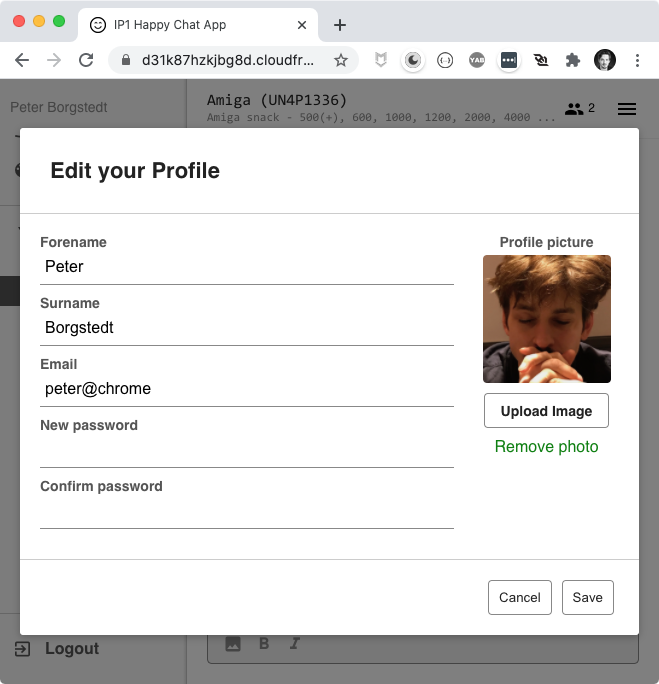
### Huvudfönster

Vid en lyckad inloggning så landar användaren in i huvudfönstret vilket består av en delad vy. Till höger är ett sidofält. Den övre delen ger möjligt att redigera användarprofilen samt skapa och prenumerera på kanaler. Den nedre delen visar de kanaler som användaren valt att prenumerera på. Vid klick på dessa så byts den vänstra vyns innehåll till kanalens.

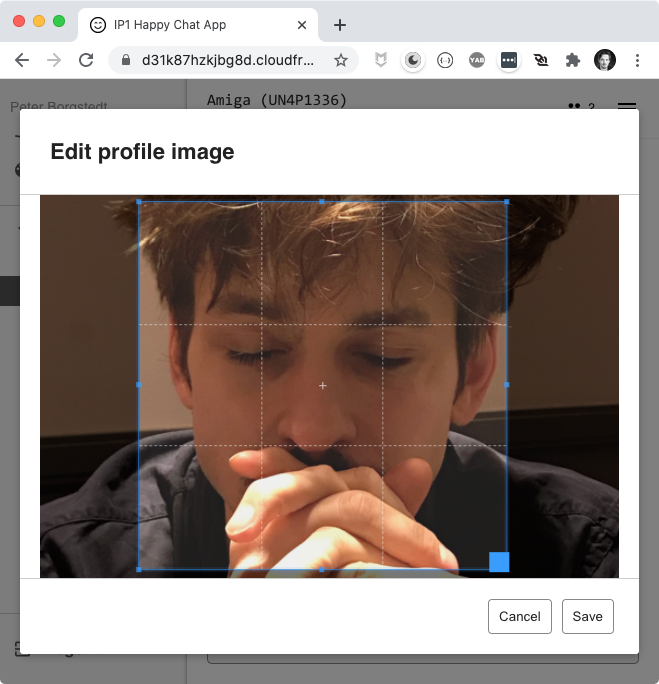


### Användarprofil

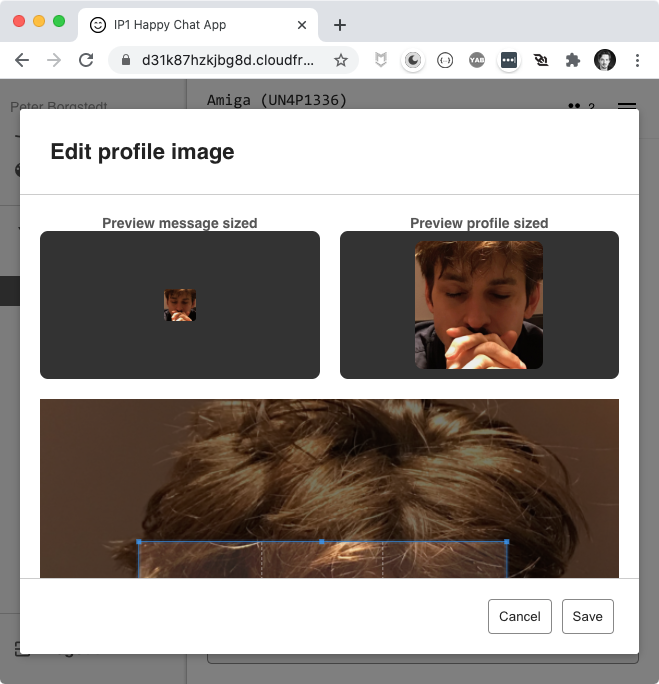
Vyn innehåller användarens detaljer och möjlighet att redigera dem.



Det går även att byta profilbild; vilket är den bild som presenteras i användarens meddelanden. Efter att en fil valts så får användaren möjlighet att skära bilden till önskat mått.

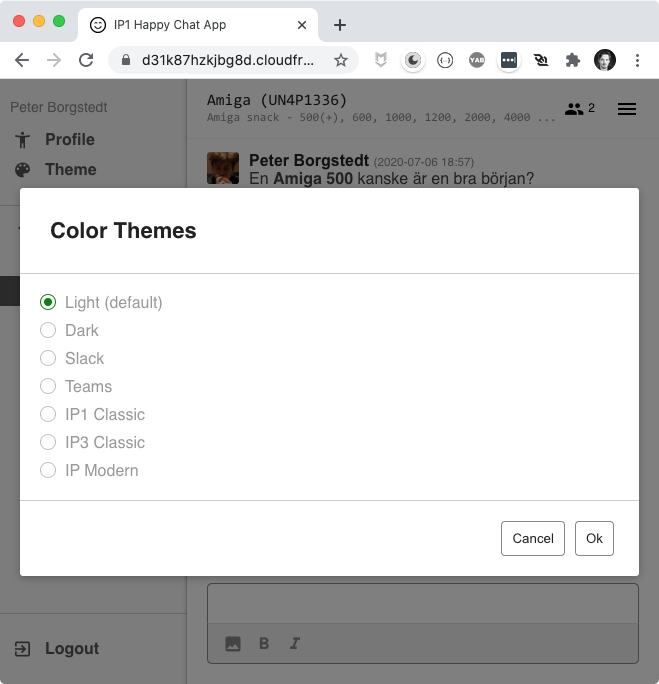


En förgranskning visar hur bilden kommer se ut i meddelanden och vid egen granskning av sin profil.



### Användarprofil

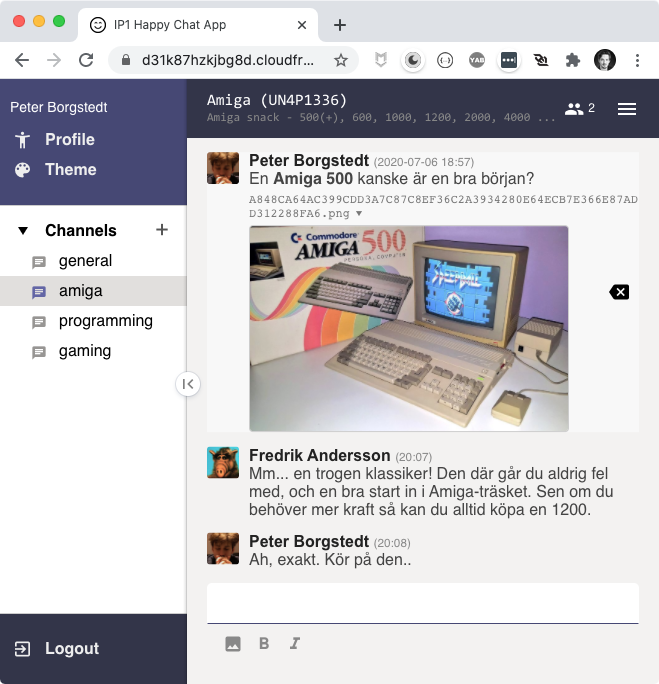
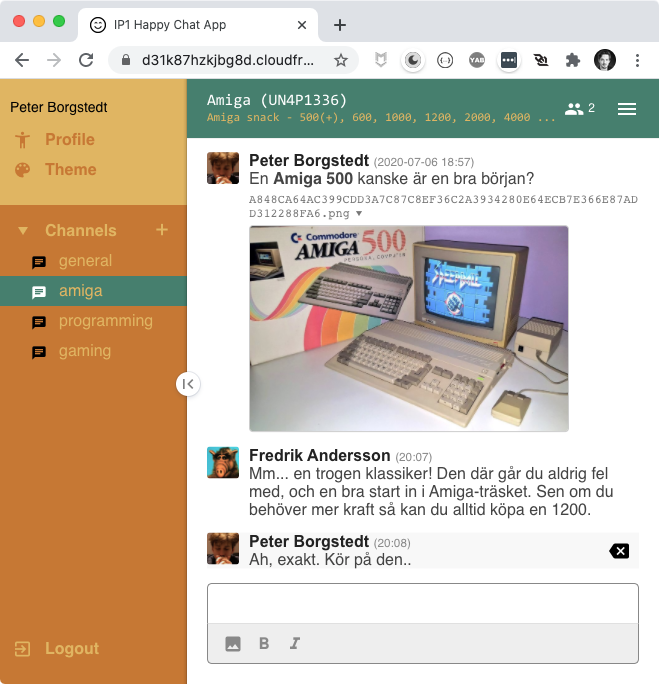
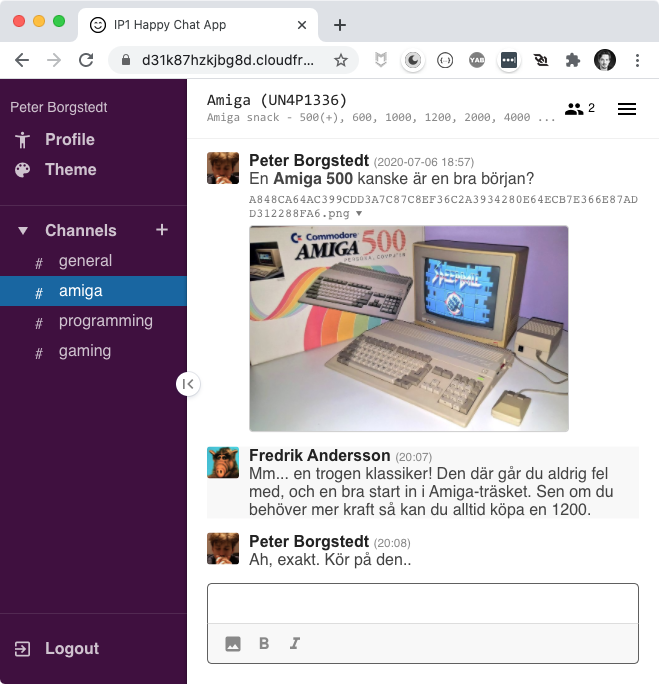
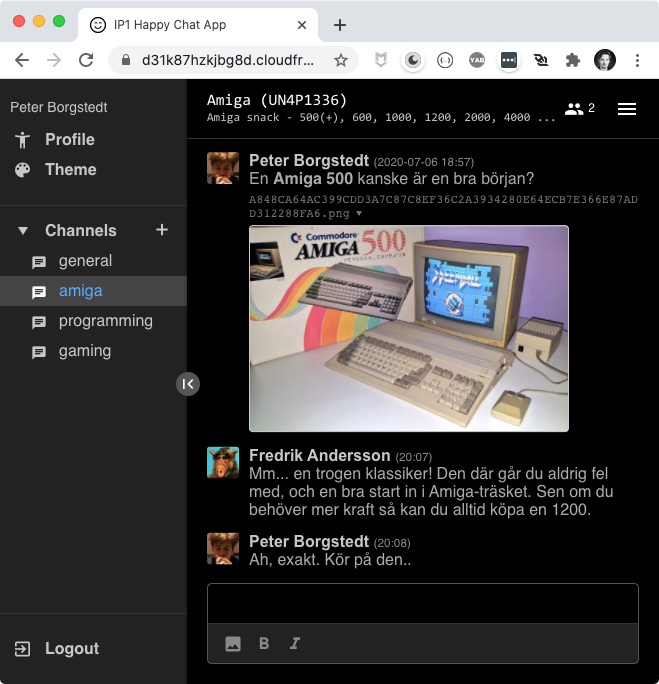
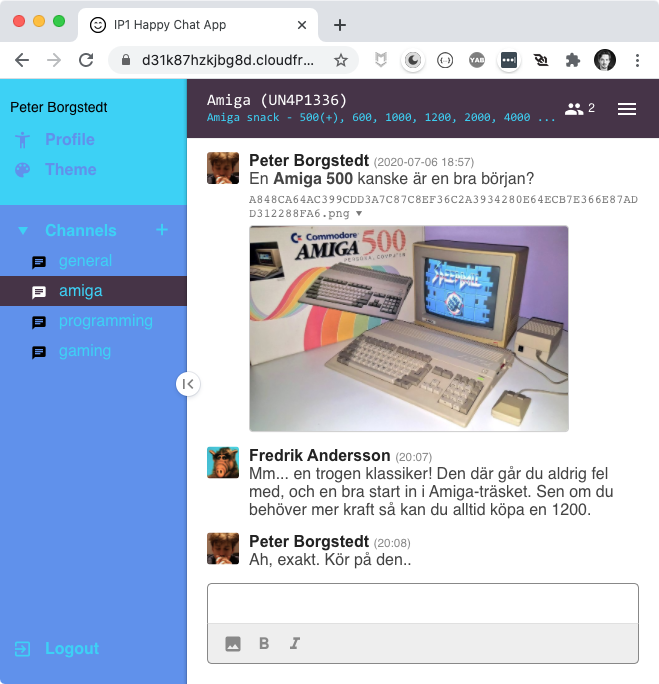
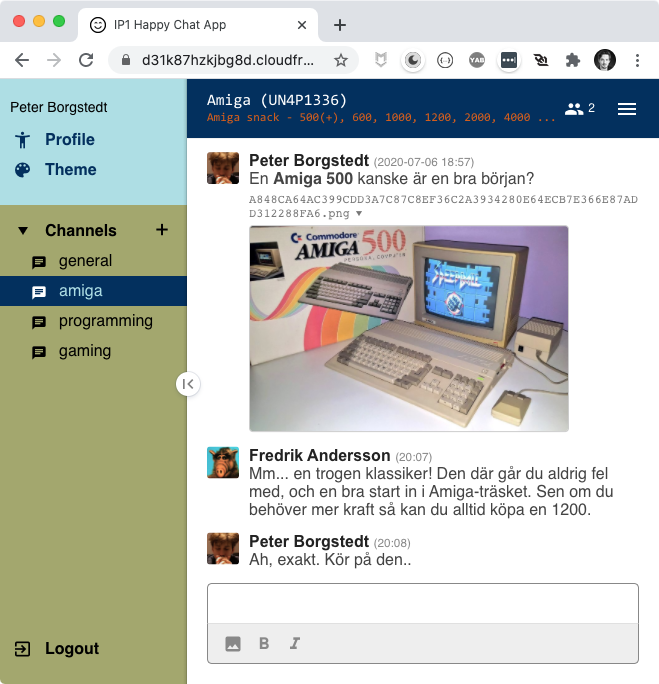
Vyn innehåller en lista med färgteman som ändrar utseendet i applikationen. Ett färgtema består av CSS variabler som byts ut mot andra.



När användaren har valt sitt tema och tryckt **OK** så sparas valet i **local-storage** och utseendet ändras.

**Slack**

**Dark**



**IP Modern**

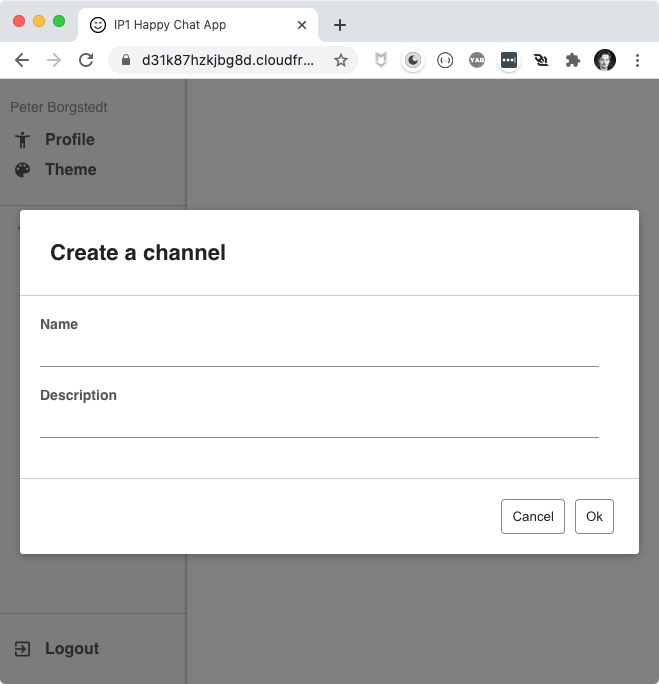
**IP3 Classic**

**IP1 Classic**

**Teams**

### Skapande av en kanal

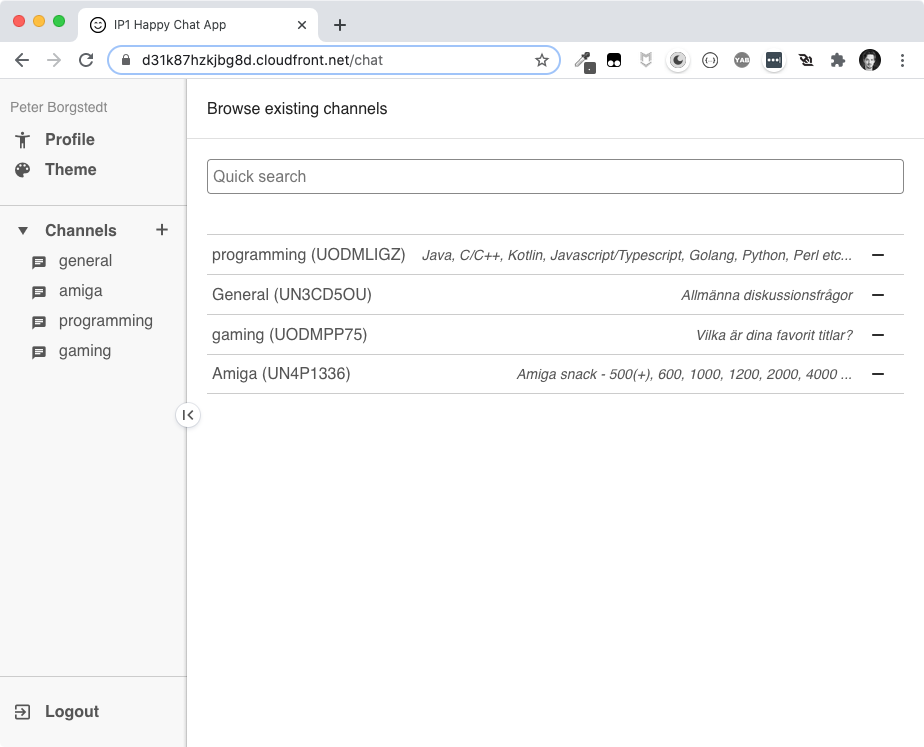
Vyn består av ett modal-fönstret där det går att ange namn och beskrivning för kanalen som sen skickas med REST till servern. Om anropet lyckas så skapas kanalen och kanalen läggs till i användarens sidofält i listan av prenumererade kanaler.



### Bläddring av kanaler

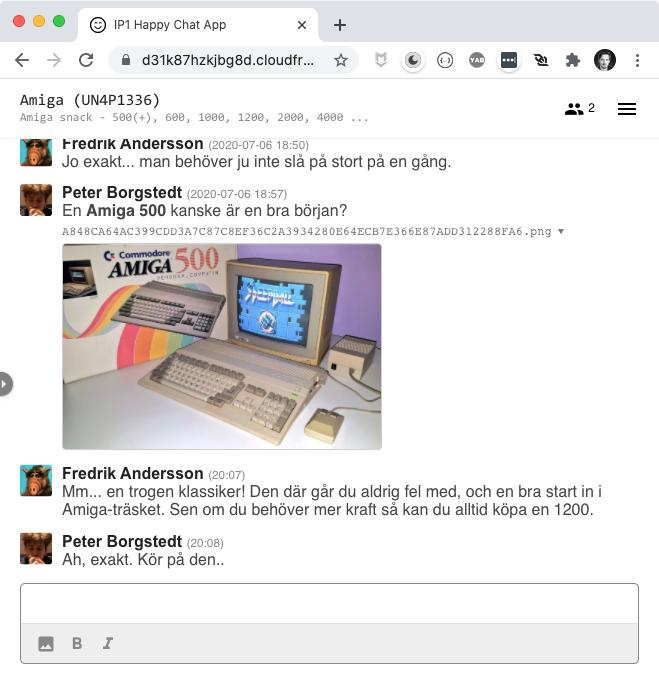
Vyn består av ett inmatningsfält och en lista med befintliga kanaler. Om det finns många kanaler så går det att filtrera via inmatningsfältet. Varje listad kanal har en ”toggle-knapp” som antingen prenumererar eller avprenumererar på kanalen. Vid prenumeration så skickas ett anrop via websocket till servern vilket lägger till användaren i kanalen. Servern skickar därefter ut ett meddelande till alla aktiva klienter som prenumererar på kanalen om att användaren lagts till, vilket kommer att inkrementera antalet abbonenter i kanalens detaljer och för användaren själv så läggs kanalen till i sidofältet.

Vid avprenumerering så utförs samma flöde men där användaren tas bort från kanalen, vilket gör att kanalen försvinner från användarens sidofält och antalet abbonenter dekrementeras i kanalens detaljer för de återstående aktiva prenumeranterna.



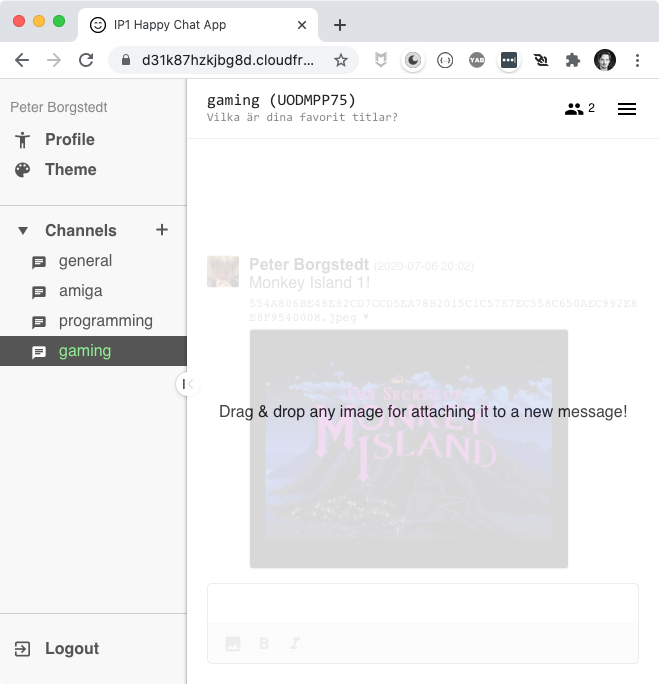
### Kanaler

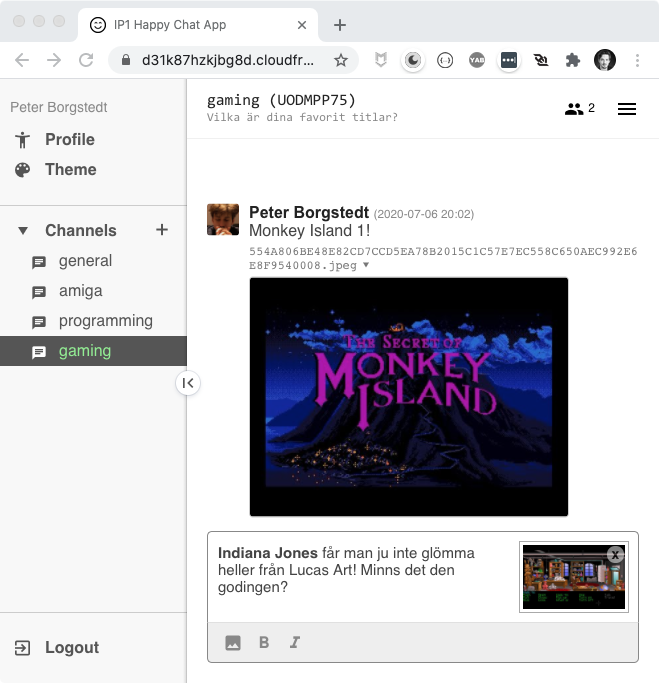
Till vänster av den delade huvudvyn presenteras innehållet för en kanal, detta sker när användaren valt att klicka på en de listande kanalerna i sidofältet.



Vyn innehåller en övre del med detaljer runt kanalen så som **namn**, **id**, **beskrivning** och antalet **prenumeranter**. Den mittersta delen innehåller meddelanden som skickats av användare, dessa kan vara av text och bild. Den nedre delen innehåller ett inmatningsfält där användaren kan skapa nya meddelande med bifogad bild. Meddelanden kan även raderas via en knapp, denna syns endast för de meddelanden som användaren själv har skrivit.

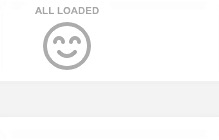
Det går även att **drag & drop** en bild in till kanalen, vilket kommer att bifoga bilden i meddelandet.





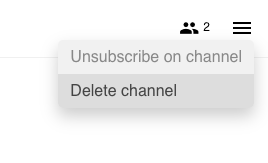
Meddelanden laddas in med paginering, därav läses en fixerad mängd in först; de senaste 10:e meddelanden. Normalt är användare intresserad av nytt innehåll och för att undvika tunga nedladdningar av tusentals meddelande för en äldre kanal så läses dem in inkrementellt när man scrollar uppåt.

Om fler meddelanden finns att ladda (vid scrollning) så kommer det indikeras när musen ligger över kanalens innehåll men ikon och text.



När alla meddelanden har laddas indikeras det med ikon och text som sakta bleknar ut.

För att ta bort ett meddelande så trycker man på knappen som visar sig när man lägger musen över ett av sina meddelanden. Knappen är lokaliserad längst till höger och det går endast att ta bort meddelanden som användaren har skapat.



För att ta bort en kanal så trycker man på ”**Delete channel**” i menyn som är lokaliserad i högra övre hörnet. Menyn har även ett alternativ för avprenumeration.  
En kanal kan endast tas bort av den användare som har skapat den. När en kanal tas bort så vidarbefodras detta till alla klienter vilka då kommer att slängas ut, kanalen, alla prenumeration och meddelanden tas bort från databasen.

## Kort beskrivning över några av tekniker som används utanför kursens innehåll

### Webkomponenter

Webkomponenter är en relativ ny teknik inom **HTML5** vilket ger ett ”native” stöd för komponentbaserad-utveckling, d.v.s. möjlighet att skapa egna HTML-element som i sin tur har ett eget scope för logik med stöd av bland annat **Shadow DOM;** vilket enklast kan förklaras som en privat **DOM** för ett element.

### REST

REST (Representational State Transfer) är en webbteknik för att överföra data mellan klient och server. Överföringen sker med JSON (JavaScript Object Notation), men även andra dataformat stöds.

För att vara tydlig på skillnaderna av de två tekniker av överföring som används så sker en transaktion av REST endast vid förfrågningar, d.v.s. klienten anropar servern med en fråga och aldrig det motsatta (då servern inte har någon aning om klienterna).

### Websocket

En websocket är en kontinuerlig anslutning mellan klient och server via en socket. Det är inget nytt fenomen, men inom webbtekniken har den varit ganska frånvarande (kanske inte helt då ett HTTP-anrop också sker via en socket, skillnaden är dock att den stängs efter att anropet har slutförts).

Via en socket kan kommunikation strömma fritt mellan klient och server vilket gör det möjligt ”pusha” ut informationen när något nytt finns snarare än att klienten ständigt behöver fråga (finns det något nytt?) vilket även kallas för ”polling”.

Vid anslutning av en websocket så har en utökad del lagts till på baksidan med autentisering, d.v.s. en webtoken behövs anges som en query-paramenter vid uppkoppling, anges ingen eller om den är felaktig så kommer ingen anslutning att ske.

### Webtoken

För att säkra upp mellan server och klient så behöver användaren intyga sin identitet, det görs med en autentisering. En användare behöver först registrera sig i systemet för att nyttja webapplikationen, därefter behövs en inloggning vilket i sin tur ger en signerad token som bekräftar användarens identitet. Denna token används vid kommunikation med servern. Registrering och inloggning utförs med HTTP-anrop via rest mot det API som servern tillhandahåller. Token verifieras på baksidan för REST och websocket (handshake).

### CSS Variabler

Med variabler som stöd i CSS så går det enkelt att byta ut befintliga värden i CSS-scheman med kod och dynamiskt byta ut färger och annat utan större ansträngningar. Detta är vad som används för att ändra ”look and feel” (färgteman) i applikationer.

### Summering

Det har varit ett jättekul projekt och speciellt eftersom man kunna kombinera den tillsammans med IP1, vilket gjorde att jag kunde bygga något litet extra. Det blev sommarens stora sysselsättning tillsammans med de övriga uppgifter som kursen innehöll. Tungt och stundvis omfattande men roligt och skönt nu i efterhand att se vad man har lyckas åstadkomma.

Om jag skulle göra om det så finns det nog massor med saker jag troligen skulle gjort annorlunda, vilket iofs är sant med det mesta av saker man gör. Huruvida jag skulle bygga någon större applikation med **Webkomponenter** är jag osäker om då det kräver alldeles för mycket kod och tid. Med ett ramverk som **VueJS** får man så mycket gratis och man kan fokusera mer på vad man ska bygga än grundläggande mekanismer som borde finnas på plats.

**Websocket** är fantastiskt för **push** av data snarare än **poll**, dock ersätter den inte fullt ut **REST** som lämpar sig bra för data som inte är kritisk närvarande för en användare.

Det var många bitar som jag hade velat få på plats, som exempel stöd för video med webkamera vilket jag i sådant fall hade strömmat som datagram (UDP) då TCP inte hade varit lämplig p.g.a. paket-validering som skulle bli för långsamt, jag behövde lägga min nivån någonstans (och kanske är det något som jag kan göra på fritiden). Samma gäller offline-stöd och mobilanpassning; idag har applikationen responsivt stöd för desktop och plattor; d.v.s. den är responsiv och det finns till och med anpassning för mobiler i ett fåtal vyer; inloggning och registrering.

Det hade varit häftigt att få till en progressiv webapplikation byggd helt ”native” utan ramverk, men det får bli en annan dag.  
  
Jag hoppas att mitt arbete blir uppskattat och kanske presenterar någon nytt som ingen ännu har presenterat i sin gesäll.

Övrig teknisk dokumentation finns detaljerat i koden, bl.a. hur jag resonerat och eventuella källor.