

# Accenture e-sport

---

En turneringsportal av:

Daniel Roger Hansen

Paul Mathias Høglend

Herman Vognild Rustad

Bachelor OsloMet Gruppe 10

Våren 2021



Institutt for Informasjonsteknologi  
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo  
Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR.  
10-2021

TILGJENGELIGHET  
Offentlig

Telefon: 22 45 32 00

# BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL Accenture e-sport En turneringsportal	DATO 26.05.2021
	ANTALL SIDER / BILAG 173
PROSJEKTDELTAKERE Daniel Roger Hansen Paul Mathias Høglend Herman Vognild Rustad	INTERN VEILEDER Henrik Lieng

OPPDRAKGIVER Accenture	KONTAKTPERSON Daniel Meinecke
---------------------------	----------------------------------

## SAMMENDRAG

App for Accenture e-sport er et bachelorprosjekt gjort i samarbeid med Accenture Norge. Det er en portal for å holde turneringer i dataspill. Portalen skal videreutvikles av et team etter endt bachelorperiode.

Appen er en webapp laget med .NET Core backend og React frontend. Webappen er deployet med Azure app service med en Azure SQL database og en Azure blob storage.

I webappen kan brukere:

- slette turneringer og brukere (systemadministratører),
- opprette og endre turneringer (turneringsadministratører),
- invitere brukere til lag og melde dem på «single elimination»- turneringer (lagleder),
- opprette lag, administrere brukerprofil, (innloggede brukere),
- registrere brukerprofil, se en turneringsoversikt og søke etter brukere, lag og turneringer.

3 STIKKORD
React
.NET Core
Turneringsportal

# 1 Forord

Denne rapporten beskriver arbeidet med bachelorprosjektet til gruppe 10 ved OsloMet våren 2021, gjort i samarbeid med Accenture Norge. Bachelorprosjektet handler om å lage en løsning for å arrangere turneringer i dataspill.

Rapporten består av fem deler:

- Prosjektpresentasjon
- Prosessdokumentasjon
- Produktdokumentasjona
  - Funksjonalitet
  - Programmets oppbygging og virkemåte
  - Testdokumentasjon
- Brukermanual
- Vedlegg, ordliste og referanser

De to første delene er ment for alle. De beskriver hva vi har jobbet med og hvordan vi har jobbet. De to neste delene er ment for utviklere og personer med teknisk kompetanse. Dette gjelder særlig delene «Programmets oppbygging og virkemåte» og «Testdokumentasjon».

Vi vil rette en stor takk til våre veiledere Elisabeth B. Kaarud, Jens Omfjord og Sindre Røkke hos Accenture. De har vært utrolig hjelpsomme, støttende og tilgjengelige gjennom hele bachelorperioden. I tillegg har deres interesse for prosjektet gjort det enda mer givende å jobbe med. Midtveis i prosjektet kunne produkter Sindre

Røkke fortelle at webappen skal tas i bruk og videreutvikles etter endt bachelorperiode. Det har gitt ytterligere motivasjon til dette prosjektet.

Arbeidsspråket vårt har vært engelsk og alle ressurser vi bruker er på engelsk. Gruppen har valgt å skrive denne rapporten på norsk, men det er mange engelske ord som er krevende å finne gode oversettelser for. Språkfokuset i denne rapporten blir derfor på konsekvent bruk av ord og begreper, ikke nødvendigvis på fornorskning av alle ord og begreper.

## 2 Figurliste

Figur 4-1 Daniel Roger Hansen	Figur 4-2 Paul Mathias Høglend
Figur 4-3 Herman Vognild Rustad .....	2
Figur 5-1: Mural Board med oppgavene brainstorm og clustering .....	8
Figur 5-2 Creative matrix oppgave .....	9
Figur 5-3 Importance/difficulty matrix oppgave .....	10
Figur 5-4 Design for registreringsside .....	11
Figur 5-5 Design for turneringsoversikt.....	11
Figur 5-6 Gantt-diagram .....	13
Figur 5-7 Revidert ER-Modell .....	15
Figur 5-8 Førsteutkast ER-modell.....	15
Figur 5-9 Siste sprint i Jira .....	15
Figur 5-10 CSS-variabler.....	24
Figur 5-11 Kode før formatering .....	25
Figur 5-12 Kode etter formatering.....	25
Figur 5-13 Factory for å produsere en liste med ProfileCardModel .....	26
Figur 6-1 Registreringsskjema fylt inn med brukerinformasjon .....	30
Figur 6-2 Innloggingsskjema fylt inn med brukerinformasjon.....	31
Figur 6-3 Send aktiveringskode per e-post.....	32

Figur 6-4 Oppdatere passord.....	33
Figur 6-5 Uredigert brukerprofil.....	34
Figur 6-6 Beskjæringsverktøy for profilbilde og banner .....	35
Figur 6-7 Lukket modal-vindu for oppdatering av brukerinformasjon .....	36
Figur 6-8 Modal-vindu hvor bruker ønsker å oppdatere passord .....	37
Figur 6-9 Modal-vindu hvor bruker ønsker å oppdatere brukernavn eller epost .....	37
Figur 6-10 Brukerprofil koblet opp mot Steam-konto .....	38
Figur 6-12 Steam autentiseringsflyt .....	40
Figur 6-11 Modal-vindu med mulighet til å legge til eksterne spillprofiler.....	40
Figur 6-13 Brukerprofil hvor alle redigeringsknappen er tatt i bruk .....	41
Figur 6-14 Bruker kan opprette lag på to måter .....	42
Figur 6-15 bruker prøver å opprette lag med opptatt navn gjennom brukerprofil.....	42
Figur 6-16 Bruker prøver å opprette lag med opptatt navn gjennom navbarknappen	43
Figur 6-17 Uredigert lagprofil .....	44
Figur 6-18 Modal-vindu for å søke opp spiller. Bare s333733 vises da s333769 er leder for laget og s333748 allerede er medlem. ....	45
Figur 6-19 En lagleder, et medlem og en invitert bruker på lagprofilen .....	46
Figur 6-20 Lagprofil med bilder, medlemmer, brukerinvitasjon og kamphistorikk ....	47
Figur 6-21 Første steg i prosessen av å opprette en turnering, og plassering av "Create tournament" knappen .....	49

Figur 6-22 Andre steg i turneringsprosessen.....	50
Figur 6-23 "Tooltip" med forklarende tekst, ment for nye brukere.....	51
Figur 6-24 Verktøy som lar en bruker justere best av formatet gjennom turneringen	51
Figur 6-25 Best av en frem til kvartfinalen, best av tre fra semifinale til turneringen er over.....	52
Figur 6-26 Best av en fra første kamp til kvartfinale, best av tre fra kvartfinale til finalen, og best av frem i finalen.....	52
Figur 6-27 Bruker har ikke invitert noen administrator .....	53
Figur 6-28 Bruker har invitert en administrator .....	53
Figur 6-29 Turneringskvittering.....	54
Figur 6-30 Turneringsoversikten som viser kamper som fortsatt har åpen registrering .....	56
Figur 6-31 Modal-vindu for å bli med på turnering .....	56
Figur 6-32 Turneringsoversikt filtrert for pågående kamper .....	57
Figur 6-33 Navbar tilhørende turnering med status høyre enn «0», eller maks antall lag påmeldt.....	58
Figur 6-34 Navbar tilhørende turnering med status «0», og med åpne plasser .....	58
Figur 6-35 Navbar tilhørende turnering med status «2» .....	58
Figur 6-36 Turneringens landingsside .....	59
Figur 6-37 Turneringstreet .....	60

Figur 6-38 Turneringstreet etter å ha blitt flyttet.....	61
Figur 6-39 Hvordan man beveger på turneringstreet.....	61
Figur 6-40 Turneringstreets funksjonsknapper .....	62
Figur 6-41 Turneringstreet i egen fane .....	62
Figur 6-42 Turneringstre som har zoomet inn.....	64
Figur 6-43 Turneringstre som har zoomet ut .....	64
Figur 6-44 Vindu hvor en bruker kan kontakte turneringsadministratorene .....	65
Figur 6-45 Funksjonsknappene for turneringsadministratoren, før påmelding er lukket .....	66
Figur 6-46 Funksjonsknappene for turneringsadministratoren etter påmelding er lukket .....	66
Figur 6-47 Funksjonsknapper for turneringsadministratoren etter at turnering er startet .....	66
Figur 6-48 Funksjonsknapper på en turnerings landingsside for turneringsadministratorer.....	67
Figur 6-49 En bruker får et varsel .....	68
Figur 6-50 Tom varselliste .....	68
Figur 6-51 Eksempel på et informasjonsvarsel.....	68
Figur 6-52 Invitasjonsvarsel til laget NIP.....	69
Figur 6-53: Kampron for vanlig bruker med status «0».....	70

Figur 6-54 Kamprom for vanlig bruker med status "1". «Live» melding har blitt lagt til for å vise at kampen har startet .....	71
Figur 6-55 Kamprom for vanlig bruker med status "2". Resultatet vises, vinner symboliseres med grønt, taper med rødt.....	71
Figur 6-56 chatboks .....	72
Figur 6-57 Kamprom med status "0" hvor motstanderlaget har gjort seg klare.....	73
Figur 6-58 "Ready up" når den blir trykket på .....	73
Figur 6-60 En turneringsadministrator får varsel om at en kamp skal starte.....	74
Figur 6-59 Begge lag har gjort seg klare som vist i chat, og "Ready up" knappen er blitt til en "Submit result" knapp.....	74
Figur 6-61 Modal-vindu for å melde resultat .....	75
Figur 6-63 Kampdeltaker har fått varsel om at et resultat har blitt levert .....	76
Figur 6-62 Detaljer forrige bilde .....	76
Figur 6-64 Resultat har blitt protestert .....	76
Figur 6-65 Turneringsadministrator får informasjonsvarsel om at et resultat har blitt protestert .....	77
Figur 6-66 Knappen en turneringsadministrator bruker for å overskrive et resultat ..	78
Figur 6-67 Inputfeltet for søk.....	79
Figur 6-68 Eksempel på søkeresultat med "sherman" som søketekst.....	79
Figur 6-69 Utlisting av brukere med sletting implementert.....	80

Figur 6-70 Utlisting av alle lag uten sletting implementert.....	81
Figur 6-71 Utlisting av alle turneringer, med sletting implementert .....	81
Figur 6-72: Kamprom på mobil del 1 .....	83
Figur 6-73: Kamprom på mobil del 2 .....	83
Figur 6-74: Kamprom på mobil del 3 .....	83
Figur 6-75: Sign in på mobil.....	83
Figur 6-76: Turneringsutlisting på mobil.....	83
Figur 6-77: Hamburgermeny på mobil .....	83
Figur 6-78 MVC .....	85
Figur 6-79 Database access layer .....	87
Figur 6-80: API-endepunkt i controller .....	88
Figur 6-81: Bruker Axios til å utføre en postforespørsel .....	88
Figur 6-82 Modell-objekt som matcher JavaScript-obektet sendt inn med postforespørsel.....	89
Figur 6-83: JavaScript-objekt opprettes på klienten og blir sendt med post-forespørsel .....	89
Figur 6-84: API-endepunkt i en controller .....	89
Figur 6-85 Logger feilmelding .....	90
Figur 6-86 Eksempel på bruk av asynkron programmering .....	90
Figur 6-87: Visualisering av komponenter, vist med forskjellige farger .....	93

Figur 6-88 state i klassekomponent.....	94
Figur 6-89 props i funksjonell komponent.....	95
Figur 6-90 ClientApp mappestruktur.....	96
Figur 6-91 Schema.js.....	96
Figur 6-92: Matrise med kampobjekter, visualisert som et turneringstre .....	97
Figur 6-94 Identifikasjon for NewTournament-endepunkt.....	98
Figur 6-95 Identifikasjon for Join-endepunkt.....	98
Figur 6-96 Identifikasjon for CloseTournament-endepunkt.....	99
Figur 6-97 Operasjonene vi skal se nøyere på .....	99
Figur 6-98 metode generateTournamentBlueprint.....	100
Figur 6-99 metoden SpotsInFirstRound.....	100
Figur 6-100 metoden numberOfRounds .....	101
Figur 4-6-101 Rundeindeks visualisering.....	101
Figur 6-102 metoden «AddTournamentBlueprint».....	102
Figur 6-103 metoden «ScrambleFirstRound» .....	102
Figur 6-104 Turnering med 8 lag .....	103
Figur 6-105 Turnering med 6 lag .....	103
Figur 6-106 Turnering med 5 lag .....	103
Figur 6-107 Algoritme for å fordele lag i første runde .....	104

Figur 6-108 Metoden som håndterer kamper i første runde som vinner på walkover .....	105
Figur 6-109 Identifikasjon for StartTournament-endepunkt.....	105
Figur 6-110 Identifikasjon for SubmitResult-endepunkt .....	106
Figur 6-111 Rundeindeks fra runde1 til runde 2 .....	106
Figur 6-112 Forrige bilde med lag.....	107
Figur 6-113: Klient kode: En klient starter en åpen tilkobling til serveren gjennom en hub .....	108
Figur 6-114 Metode "SendMessage" i en hub .....	108
Figur 6-115 Klienten som lytter etter en melding sendt over tilkoblingen.....	109
Figur 6-116 Implementering av hub-interface i en controller.....	110
Figur 6-117 Alle brukere har en liste med notifikasjonsgrupper i db .....	111
Figur 6-118 Notifikasjonsgruppe inneholder Id og navn .....	111
Figur 6-119 Ved oppretting av bruker blir det lagt til en notifikasjonsgruppe med navn likt som brukerens ID .....	111
Figur 6-120 Alle brukerens notifikasjonsgrupper blir gått gjennom, og klienten blir lagt til SignalR-gruppe med samme navn som notifikasjonsgruppen .....	112
Figur 6-121 Metode på serversiden som sender notifikasjon til gruppe.....	113
Figur 6-122 Klienten som lytter etter meldingen "Notification", og oppdater siden om dette blir plukket opp .....	113
Figur 6-123 Brukerprofil med varselsboks.....	114

Figur 6-124 Inputvalidering i brukergrensesnittet.....	115
Figur 6-125 Inputvalidering på server .....	115
Figur 6-126 Algoritmen for å generere et salt .....	117
Figur 6-127 Krypteringsalgoritmen .....	117
Figur 6-128 Sjekk for om brukeren er innlogget.....	118
Figur 6-129 Kode for å autentisere en bruker som turneringsadministrator.....	118
Figur 6-130 Kode for å sjekke om en av brukerne i en liste har en gitt ID .....	119
Figur 6-131 Azure resource group illustrasjon (Tfitzmac et al.) .....	120
Figur 6-132 Utklipp av vår Azure Resource Group .....	120
Figur 6-133 Azure pipelines som viser kjøringer når det er pushet kode til main-branchen.....	121
Figur 6-134 Publisere til Azure app service steg 1 .....	122
Figur 6-135 Publisere til Azure app service steg 3 .....	122
Figur 6-136 Publisere til Azure app service steg 2 .....	122
Figur 7-1 Konsollvinduet hvis bruker ikke eksisterer.....	134
Figur 7-2 Respons med brukerID, etter vellykket registrering.....	135
Figur 7-3 Payload med registreringsinformasjon .....	135
Figur 7-4 Request ULR og statuskode 200 (Success).....	135
Figur 7-5 Statuskode 400 (Bad Request) .....	135

Figur 7-6 Respons etter feilet registrering med allerede registrert e-post.....	135
Figur 7-7 Respons etter feilet registrering med allerede registrert brukernavn .....	135
Figur 7-8 Lighthouse-analyse av nettsiden (PC).....	137
Figur 7-9 Lighthouse-analyse av nettsiden (mobil) .....	137
Figur 11-1 Iterasjon to av oppgavene brainstorm og clustering .....	151
Figur 11-2 Iterasjon to av oppgaven creative matrix.....	151
Figur 11-3 Iterasjon to av oppgaven Importance/difficulty matrix.....	152

### **3 Innholdsfortegnelse**

1 Forord .....	iii
2 Figurliste .....	v
3 Innholdsfortegnelse .....	xv
4 Prosjektpresentasjon .....	1
4.1 Introduksjon .....	1
4.2 Presentasjon av gruppen .....	2
4.3 Presentasjon av veiledere .....	3
4.4 Oppgave .....	3
4.5 Teknologi og verktøy .....	4
4.5.1 Verktøy .....	4
4.5.2 Teknologi .....	5
5 Prosessdokumentasjon .....	6
5.1 Forord .....	6
5.2 Oppstart .....	6
5.3 Planlegging .....	7
5.3.1 Form Workshop .....	7
5.3.2 Brainstorm .....	7

5.3.3 Design av brukergrensesnitt.....	10
5.3.4 Styringsgruppemøte.....	12
5.3.5 Fremdriftsplan .....	13
5.3.6 Databasemodellering .....	14
5.4 Utviklingsprosessen .....	15
5.4.1 Arbeidsfordelingen .....	15
5.5 Statusmøter.....	16
5.5.1 Internveiledning.....	16
5.6 Sprintene .....	17
5.6.1 Sprint 1.....	17
5.6.2 Sprint 2.....	18
5.6.3 Sprint 3.....	19
5.6.4 Sprint 4.....	19
5.6.5 Sprint 5.....	20
5.6.6 Sprint 6.....	20
5.6.7 Sprint 7.....	21
5.6.8 Produkteliers ønsker .....	22
5.7 Utfordringer .....	23
5.7.1 Hosting på Azure.....	23

5.7.2 Discord.....	23
5.8 Nyttige triks i utviklingen.....	23
5.8.1 CSS-variabler.....	23
5.8.2 Prettier - Code formatter .....	24
5.8.3 Factories .....	25
5.8.4 Seeding av database .....	26
5.9 Refleksjon.....	26
6 Produktdokumentasjon.....	29
6.1 Forord.....	29
6.2 Kort beskrivelse av produktet .....	29
6.3 Hovedfunksjonalitet .....	29
6.3.1 Registrering og innlogging .....	29
6.3.2 Brukerprofil.....	33
6.3.3 Oppretting av lag.....	41
6.3.4 Lagprofil .....	43
6.3.5 Oppretting av turnering .....	48
6.3.6 Turneringsoversikt.....	55
6.3.7 Turneringsside .....	57
6.3.8 Turneringsside for turneringsadministrator.....	66

6.3.9 Varsel.....	67
6.3.10 Kamprom og chat.....	69
6.3.11 Søkefunksjonalitet.....	78
6.3.12 Systemadministrator .....	79
6.3.13 Andre krav fra kravspesifikasjonen .....	82
6.4 Programmets oppbygging og virkemåte .....	85
6.4.1 Systemarkitektur .....	85
6.4.2 Serverside .....	85
6.4.3 Klientside .....	91
6.4.4 Turneringslogikk.....	97
6.4.5 SignalR - Oppdatering mellom klienter i sanntid .....	107
6.4.6 Sikkerhet .....	113
6.5 Deployment og hosting.....	119
6.5.1 Microsoft Azure .....	119
6.5.2 Azure App Service – Web App.....	121
6.5.3 Azure pipelines continuous delivery .....	121
6.5.4 Visual Studio Publish .....	122
6.5.5 Database på Azure Server.....	122
7 Testdokumentasjon .....	124

7.1 Innledning .....	124
7.2 Brukertestene .....	124
7.2.1 Gjennomføring .....	126
7.2.2 Oppsummering av brukertestene .....	132
7.3 React Developer Tools .....	133
7.4 Postman .....	133
7.5 Chrome Devtools .....	133
7.5.1 Konsollvinduet .....	134
7.5.2 Nettverksfanen .....	134
7.5.3 Lighthouse .....	136
7.6 Azure app service .....	137
8 Brukermanual .....	138
8.1 Forord .....	138
8.2 Statuskoder .....	138
8.2.1 Turnering .....	138
8.2.2 Kamp .....	138
8.3 Hvordan kjøre webapplikasjonen lokalt .....	139
8.3.1 Programmer .....	139
8.3.2 Ressurser .....	139

8.4 Mangler og videre arbeid.....	140
8.4.1 Mangler .....	140
8.4.2 Videre arbeid.....	141
9 Ordliste .....	143
10 Referanser.....	144
11 Vedlegg .....	145
11.1 Oppdragsgivers vurdering .....	145
11.2 Kravspesifikasjon .....	146
11.2.1 Bakgrunn.....	146
11.2.2 Leserveileitung .....	146
11.2.3 Prosjektomfang .....	146
11.2.4 Funksjonelle krav .....	147
11.2.5 Ikke-funksjonelle krav.....	149
11.2.6 Rammebetingelser .....	150
11.3 Design thinking iterasjon 2 .....	151

# 4 Prosjektpresentasjon

## 4.1 Introduksjon

Siden august 2020 har Accenture Norge hatt et veldig lite gamingmiljø som har blitt administrert gjennom e-post og Teams. Denne gruppen heter "Spillgruppen" og er en altomfattende gruppe som dekker både kompetitiv gaming, avslappet gaming, brettspill og mye mer.

I august (2020) ble det undersøkt om det var interesse internt i selskapet for å samle folk til et lag og melde seg på en bedriftsliga i spillet Counter-Strike: Global Offensive (CS: GO). Undersøkelsen viste at det var en stor interesse og 20 ansatte ble med på dette.

Det har blitt gjennomført interne turneringer før, men disse har vært små og blitt administrert manuelt gjennom Excel. Om det skal bli arrangert noe større, så blir det store mengder manuelt arbeid og man må vurdere å leie inn noen for å ordne dette.

I slutten av oktober 2020 lagde Accenture oppgaveforslag til bachelorprosjekter i IT. Blant dem var følgende oppgave:

### App for e-sport gruppen

*Applikasjon for Accenture nystartede e-sportslag i CS. En applikasjon hvor man kan opprette turneringer for spillere og bruke APIer for å hente ut spiller statistikk.*

*Adminfunksjonalitet:*

- *Opprette lag og legge til spillere*
- *Sende ut varsel til spillere*
- *Lage turneringer*

### *Medlemsfunksjonalitet:*

- *Følge opp statistikk*
- *Delta i turneringer*
- *Profilside*

## 4.2 Presentasjon av gruppen

Gruppen består av Daniel Roger Hansen, Paul Mathias Høglend og Herman Vognild Rustad.



Figur 4-1 Daniel Roger Hansen



Figur 4-2 Paul Mathias Høglend



Figur 4-3 Herman Vognild Rustad

Daniel Roger Hansen er 22 år og har drevet med webdesign i mange år. Dermed har han et godt øye for brukervennlig design. Paul Mathias Høglend er 29 år og har en master i algebraisk tallteori fra Universitet i Oslo. Han er et skikkelig ordensmenneske og er opptatt av god prosjektorganisering. Herman Vognild Rustad er 21 år og har konkurrert i CS: GO siden 2015 med deltagelser i Telialigaens høyeste divisjon. Han har dermed bred erfaring med turneringsplattformer og vet hva som skiller de beste fra

de middelmådige. Alle gruppemedlemmene går på linjen informasjonsteknologi og har samarbeidet på mange prosjekter og innleveringer allerede siden studiestart 2018.

## 4.3 Presentasjon av veiledere

I Accenture har gruppen Elisabeth B. Karud og Jens Omfjord som veiledere. Sindre Røkke er produkteier. I tillegg er Daniel Meinecke og Marius Torsrud kontaktpersoner angående generelle spørsmål knyttet til bachelorprosjektet. Internveileder på OsloMet er Henrik Lieng.

## 4.4 Oppgave

Gruppen justerte oppgaven i henhold til oppgaveteksten med en gang. I samråd med veiledere og produkteier ble vi enige om å lage en turneringsplattform for flere typer dataspill. I tillegg ville vi ikke inkludere eksterne APIer for spillstatistikk, da dette ville bli alt for omfattende for prosjektet. Til slutt kom vi fram til at oppgaven skulle være:

Oppgaven går ut på å designe og utvikle en turneringsplattform der man kan registrere seg som bruker, bli med på lag og delta i turneringer i forskjellige dataspill. I tillegg skal man ha mulighet til å opprette turneringer.

## 4.5 Teknologi og verktøy

### 4.5.1 Verktøy

Kommunikasjon	Slack (internt i gruppen)  Microsoft Teams (kontakt med veiledere og produkteier)
Lagring og filhåndtering	Google Drive og Google Docs  Microsoft Word (sluttrapport)
Utviklingsverktøy	Visual Studio 2019  Visual Studio Code  Node.js  DB Browser for SQLite
Versjonskontroll og utviklingsstyring	Github  Jira
Design	Adobe XD
Deployment og Continuous delivery	Azure DevOps  Azure Portal (Azure app service)

## 4.5.2 Teknologi

Klientkode	React CSS
Serverkode	.NET Core (C#) SQLite (database lokalt)

# 5 Prosessdokumentasjon

## 5.1 Forord

I dette kapittelet skal vi dokumentere planleggingen og prosessen i utviklingen av webapplikasjonen vi skal lage. Når det står «veileder» refereres det til veiledere hos oppdragsgiveren Accenture. Vi har også en veileder fra OsloMet, men vil da bruke betegnelsen «internveileder».

## 5.2 Oppstart

Bachelorperioden startet offisielt 12. januar med introduksjonsmøte hos Accenture og påfølgende møte med veiledere og produkteier. Før dette hadde gruppen bestemt seg for å se på nettkurs i React for å bli godt forberedt til bachelor. Gruppen hadde gjort et kort individuelt prosjekt med React på høsten. Siden dette prosjektet bare varte i tre uker, var det i tillegg til å lære sentrale konsepter i React, veldig viktig å lære seg ES6 Javascript-konsepter, som er viktige for å kunne utvikle i React. Vi kommer tilbake til React i kapittel 6.4.3. For ES6 Javascript refererer vi til nettsiden <https://www.javascripttutorial.net/es6/>.

I starten av januar møtes gruppen på Slack. Grunnet koronapandemien møttes gruppen kun en gang fysisk. Det var den dagen gruppemedlemmene skulle hente bærbare PC-er på kontorene til Accenture på Fornebu. Alle andre møter har vært virtuelle.

Da diskuterte vi planlegging av bachelor basert på tidligere programmeringsprosjekter og faget systemutvikling. Disse diskusjonene var ikke veldig konkrete, men hjalp oss å komme litt i gang.

Vi hadde et mål om å planlegge prosjektstrukturen og databasetabeller godt, siden bacheloroppgave er et større prosjekt. Gruppen har at gode erfaring i tidligere prosjekter med å lagde serverkoden. Det gir en god oversikt over hvilke klasser som

snakker med klienten og hvilke klasser som snakker med databasen. I tillegg visste vi etter å ha jobbet med nettkurs i React, at vi burde tenke over å lage komponentene (byggeklossene), så gjenbrukbare som mulig. Vi ville gå for smidig arbeidsmetodikk fordi vi så for oss at prosjektet ville endre seg mye underveis. Den smidige arbeidsmetodikken Scrum var noe vi brukte en del tid på. Særlig dette med å bruke en Scrum-tavle.

En annen ting vi var bevisste på var dette med versjonshåndtering. Under oppstarten var vi usikre på hva slags rutine Accenture hadde for dette. Vi hadde brukt Git via Github og håndterte dette via en IDE. Uansett om vi ikke kjente til rutine til Accenture, var vi bevisste på at vi skulle bruke egne grener (branches) for å unngå feil i hovedgrenen og slippe hyppige merge conflicts. Det er utrolig vanlig og kan være veldig tidkrevende om en gruppe ikke har gode rutiner på det.

## 5.3 Planlegging

### 5.3.1 Form Workshop

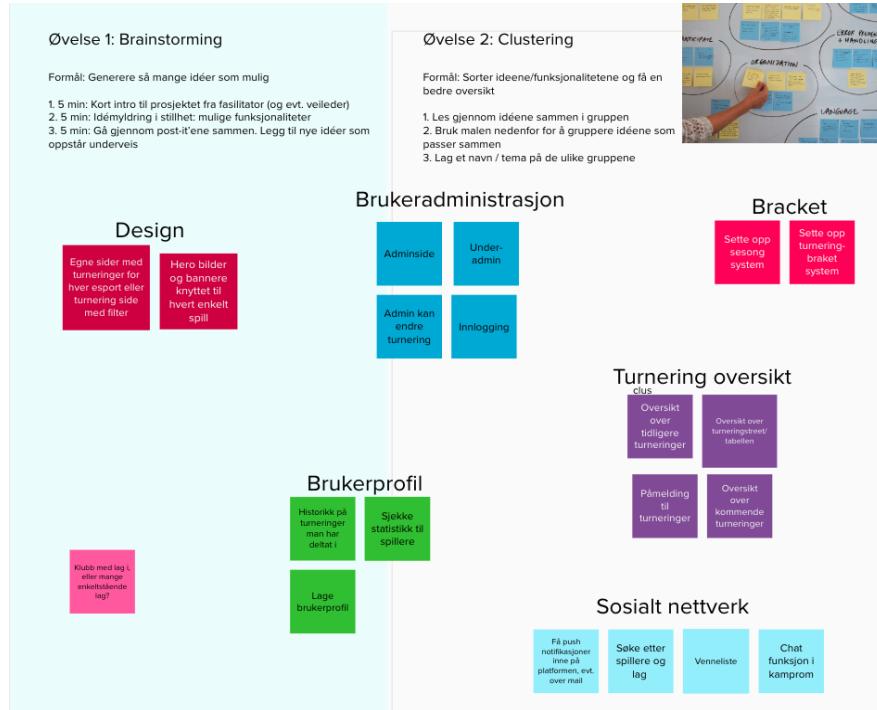
Accenture ville gi sine bachelorstudenter en god start med en workshop i FORM Design thinking. Dette skjedde torsdag 14. januar på Teams. Målet var å gi noen nye perspektiver og teknikker for å planlegge et IT-prosjekt. Hver bachelorgruppe med veiledere ble plassert i et breakout room for å løse forskjellig oppgaver i verktøyet Mural. Oppgavene i workshoppen var:

### 5.3.2 Brainstorm

Denne oppgaven går ut på å komme på så mange ideer som mulig relatert til oppgaven på kort tid. I denne sammenhengen hadde vi fem minutter. Deretter diskuterte vi i fem minutter de ideene vi kom opp med og supplerte med flere ideer.

### 5.3.2.1 Clustering

Denne oppgaven handler om å samle ideer for å få oversikt over funksjonalitetene.



Figur 5-1: Mural Board med oppgavene brainstorm og clustering

### 5.3.2.2 Creative Matrix

Denne oppgaven går ut på å sette opp en spesifikk tabell for funksjonalitet som er bestemt i forrige oppgave. Målet er å gjøre seg opp tanker rundt:

- avhengigheter til annen funksjonalitet,
- potensielle tekniske utfordringer,
- hvor mye tid tar hver enkel funksjonalitet,
- tilganger som eventuelt behøves,
- andre ting som er viktig å tenke på. (Se bildet på neste side)

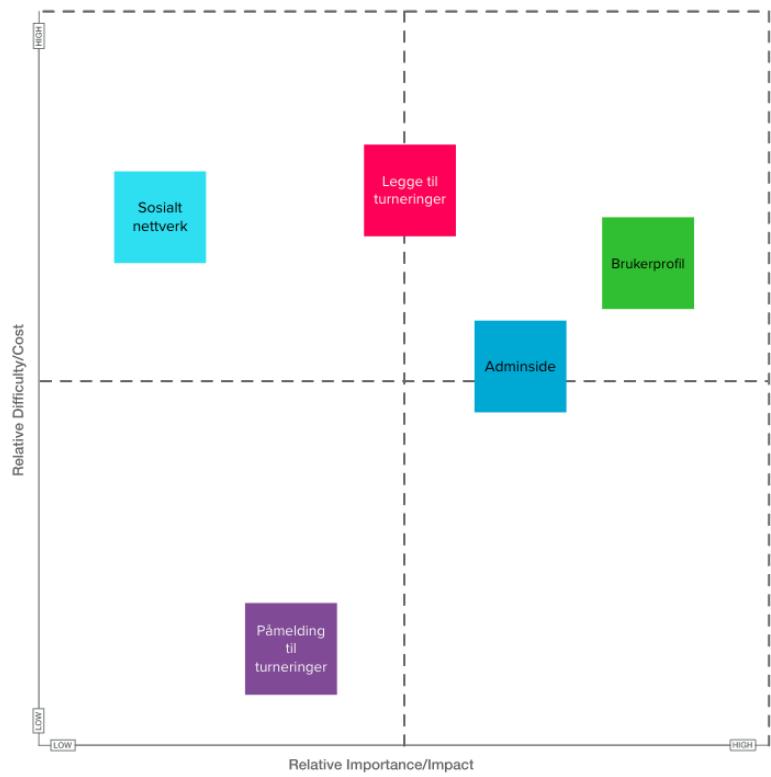
Funksjonaliteter →

	Brukerprofil			Adminside	Legge til turneringer		Påmelding til turneringer			
Avhengigheter	Innlogging	Registrering	Sikkerhet		Organisator		Overikt over turneringer	Være i et lag		
Tekniske utfordringer	Bruker profiller koblet opp mot plattformer osm steam, riot etc (langsiktig)			Håndtering av sessions Skille på ulike roller	Mulige parametere som skal inkluderes Vidg av funksjonshjemmet		Oversiktlig oversikt over turneringer			
Tid	20		20		40		20			
Tilgang (feks til personer, materiell, ++)										
Diverse										

Figur 5-2 Creative matrix oppgave

### 5.3.2.3 Importance/Difficulty matrix

Denne oppgave går ut på å vurdere innvirkning og vanskeligheten til funksjonaliteten. Det betyr at om en funksjonalitet plasseres øverst til høyre, vil den ha ekstra innvirkning og være vanskelig. Tilsvarende om en funksjonalitet plasseres nederst til venstre vil den ha mindre innvirkning og være lettere. Med en sånn matrise får gruppen et bilde på prioritering med hensyn til vanskelighetsgrad og innvirkning. (Se Figur 5-3)

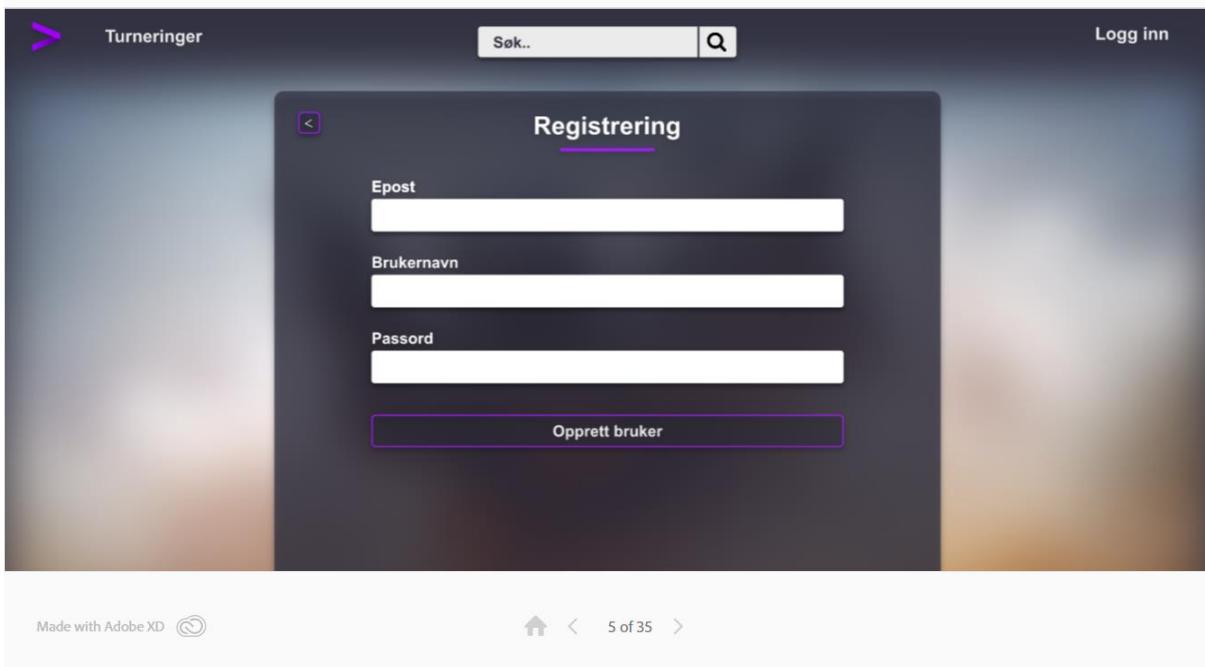


Figur 5-3 Importance/difficulty matrix oppgave

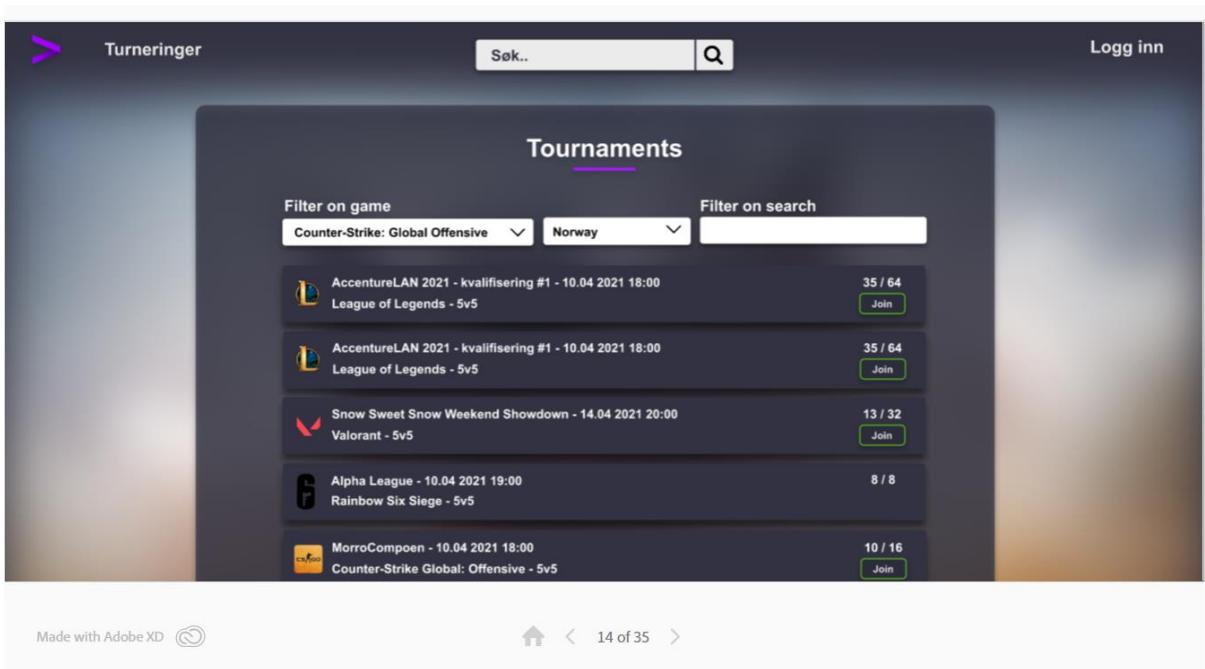
Disse planleggingsoppgavene var helt nytt for gruppen og vi tenkte at vi hadde fått litt for lite tid på alle oppgavene. Derfor kjørte vi en ny iterasjon av oppgavene. Se 11.3 for resultatene av denne iterasjonen.

### 5.3.3 Design av brukergrensesnitt

Vi brukte Adobe XD aktivt i prosessen for å designe brukergrensnittet. Dette var viktig for at produkteier kunne se hva vi skulle lage, slik at han kunne komme med innspill. Før hver sprint viste vi produkteier forslag til design. Det skjedde på våre veiledningsmøter som vi hadde hver 14. dag.



Figur 5-4 Design for registeringsside



Figur 5-5 Design for turneringsoversikt

Etter presentasjon av design-forslagene ga produkteier bekreftelse og eventuelle innspill. Med få unntak var produkteier fornøyd med det designet vi presenterte. I

starten av utviklingen var språket i webapplikasjonen på norsk, men produkteier ønsket å bytte til engelsk slik at Accenture-avdelinger i for eksempel Tyskland og Norden også kan benytte seg av denne løsningen i fremtiden.

Under utformingen av designet ønsket vi å skape troverdighet hos brukeren. Vi valgte derfor å plassere Accenture sin logo tydelig synlig øverst i venstre hjørne. Vi benyttet oss også av Accentures lilla logofarge som aksentfarge i fargepaletten. Bakgrunnsbildet er et motiv fra CS:GO, som er det mest populære spillet i spillgruppen, dette for å skape en familiær følelse hos applikasjonens målgruppe. Vi valgte å legge en blur-effekt på bildet, slik at det ikke skulle ta for mye fokus vekk fra funksjonaliteten. Designet for PC-versjonen har en sentrert boks med innhold, som ikke strekker seg ut i hele skjermbredden. Dette var et bevisst designvalg for å rette brukeren sin oppmerksomhet mot delen av siden den kan interagere med, og for å gjøre det enklere å gjøre tilpasninger for skalering til mobilskjerm.

Ved å følge linken under kan man se alt som ble designet i AdobeXD

<https://xd.adobe.com/view/223a35f2-d4e0-4993-bc9f-53681426ef1a-7769/grid>

### 5.3.4 Styringsgruppemøte

Vi hadde et styringsgruppemøte med Accenture 10. februar. Dette var en presentasjon om hva gruppe hadde planlagt. Gruppen fikk skryt for design og demo av foreløpig funksjonalitet.

Samtidig fikk gruppen tilbakemelding på følgende:

Hva er det dere lager?

Med det siktet de til tydelig kravspesifikasjon for produktet. Presentasjonen vi holdt hadde ikke inkludert en kravspesifikasjon.

Hvordan har dere planlagt utviklingen?

Vi lagde et Gantt-diagram i ettertid.

Hvilken teknologistack bruker dere?

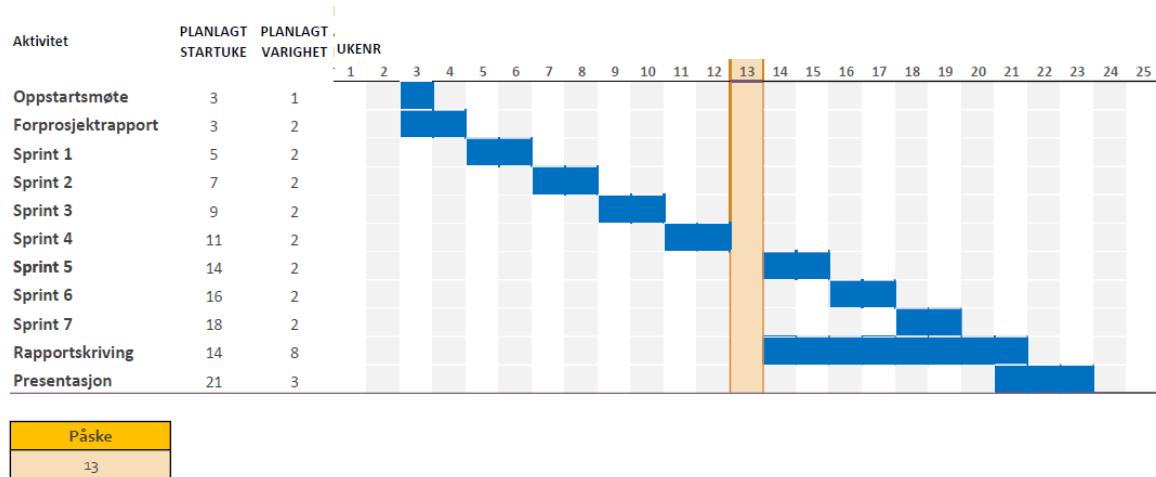
Det kom ikke klart fram at vi skulle bruke React med .NET Core.

Hvordan løser dere utfordringer?

Her hadde vi heller ikke kommunisert hvordan vi skulle løse utfordringene. Vi hadde på dette tidspunktet identifisert at e-postfunksjonalitet kunne bli en utfordring. I tillegg tenkte vi at: turneringer, chatrom, varsler og eksterne spillkontoer også ville bli krevende.

Når det gjaldt turnering tenkte vi at vi måtte planlegge godt. Vi så for oss at vi skulle fokusere på «Single elimination», også kalt cupspill. Angående e-post, chatrom, varsler, turneringstre og eksterne spillkontoer, tenkte vi at vi måtte lese oss godt opp på dokumentasjonen til eksisterende løsninger. Samtidig var vi klare på å ha god kommunikasjon og gi beskjed om de utfordringene vi måtte møte på. Vi tenkte at hjemmekontor skulle gå fint. Vi har jobbet godt sammen over videosamtaler helt siden 13. mars 2020.

### 5.3.5 Fremdriftsplan



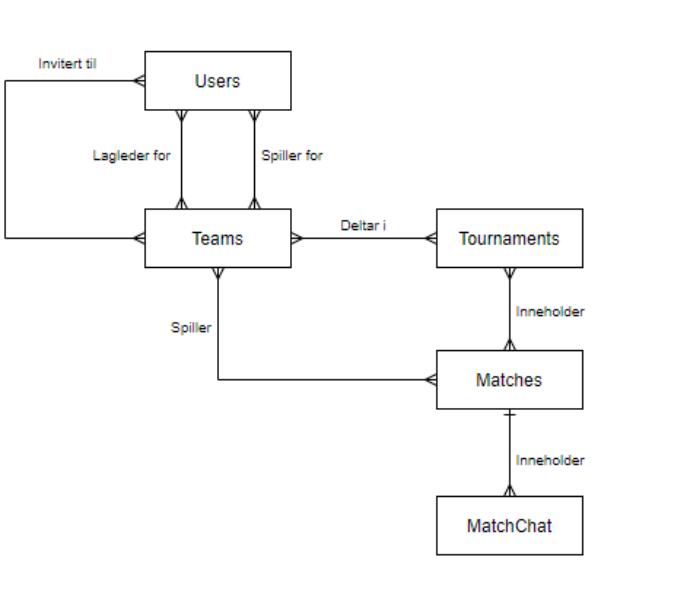
Figur 5-6 Gantt-diagram

Da vi opprettet Gantt-diagrammet var det viktig å sette av nok tid til rapportskriving. Vi tenkte også at vi måtte avslutte utviklingen av ny funksjonalitet noen uker før innleveringsfrist.

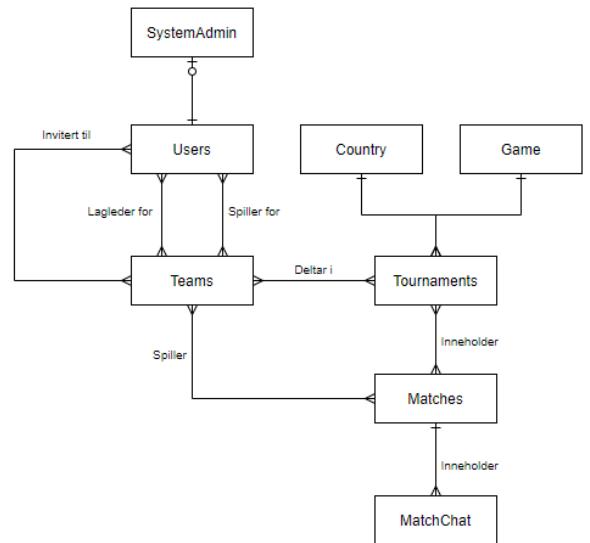
### 5.3.6 Databasemodellering

For å gi oss en oversikt over hva som trenger å bli lagret i en database og hvilke avhengigheter disse dataene har, lagde vi et ER-diagram. Dette ble gjort ganske tidlig etter Design thinking workshopen, og før noe av utviklingen startet. Med et ER-diagram ble det lettere å avdekke eventuelle mangler og svakheter i databasen før vi startet med implementasjonen. Det hjalp oss også med å konkretisere et system som føltes veldig abstrakt på tidspunktet diagrammet ble opprettet. Det bidro også til å skape en felles forståelse av databasestrukturen for alle gruppemedlemmene, noe som bidro til å redusere forvirring og misforståelser innad i gruppen. For å lage diagrammet benyttet vi oss av verktøyet Draw.io (<https://app.diagrams.net/>). Før utvikling startet, så modellen ut som på Figur 5-8.

Etter hvert som prosjektet skalerte, og vi fikk nye og uforutsette oppgaver av produkteier ble vi nødt til å gjøre noen endringer i databasemodellen vår. En del av disse endringene kom som følge av at webapplikasjonen ikke lenger bare var ment for spillgruppen til Accenture nasjonalt, men for bedriften på globalt nivå. Det var derfor ønskelig å knytte sammen turnering og land for å ha mulighet til å filtrere slik at en bruker bare ser turnering fra ønsket land. I tillegg ble det etterspurt mulighet til å legge begrensning på at det bare skal være mulig for lag fra bestemte land å delta på spesifikke turneringer.



Figur 5-8 Førsteutkast ER-modell



Figur 5-7 Revidert ER-Modell

## 5.4 Utviklingsprosessen

### 5.4.1 Arbeidsfordelingen

Vi brukte Jira til å fordele oppgaver. Jira ble integrert i kommunikasjonsverktøyet Slack, slik at vi enkelt fikk beskjed når oppgaver ble opprettet, påbegynt eller fullført.

Jira-skriftskjerm fra "ESG Sprint 7" viser følgende oppgaver:

- ESG-199 | GJØREMÅL**: 2 underoppgaver En bruker skal kunne chatte i kamprom
- ESG-216 | GJØREMÅL**: 5 underoppgaver En bruker skal kunne melde seg på en turnering
- ESG-217 | UNDER ARBEID**: 2 underoppgaver En turneringsadmin skal kunne endre turneringsinfo
- Fjerne admin med knapp**: ESG-242
- Backend-kode**: ESG-251
- ESG-169 | GJØREMÅL**: 3 underoppgaver Rydde opp i CSS
- ESG-229 | GJØREMÅL**: 3 underoppgaver En bruker skal kunne motta notifikasjoner i navbar

Figur 5-9 Siste sprint i Jira

Dette er en type Scrum-tavle slik vi hadde planlagt for arbeidsfordeling.

Vi hadde «stand up»-møter annenhver dag. I disse møtene fortalte hvert gruppemedlem hva de hadde gjort og hva de skulle gjøre. Vi valgte «stand up»-møter annenhver dag på grunn av at vi hadde fag ved siden av bachelor. I tillegg hadde et av gruppemedlemmene deltidjobb hver torsdag. Dessuten tenkte vi at om hver enkelt hadde litt ekstra tid på å jobbe med brukerhistorier, ville vedkommende få gjort mer. Samtidig tok vi ofte kontakt om større utfordringer i arbeidet.

## 5.5 Statusmøter

Hver 14. dag hadde vi statusmøter med veiledere og produkteier. Vi startet med å vise frem demo av hvordan appen så ut etter endt sprint. Så viste vi frem designet vi hadde laget i AdobeXD, ment for neste sprint. Etter godkjenning og innspill på design, gikk vi videre til å diskutere prioriteringer før neste sprint. Sprint 3 ble for eksempel en sprint for å restrukturere mappestrukturen på klientsiden og få daværende funksjonalitet ferdig.

På statusmøte 12. februar ga veilederne våre oss en innføring i «sprint retrospekt». Da skrev vi ned: det som hjelper oss i utviklingen, hva som holder oss tilbake og hva vi skulle gjøre mer av. Her var gruppen enig i alle punkter. Siden vi hadde god kommunikasjon og hadde mye å gjøre, valgte vi ikke å gjøre flere «sprint retrospektiv».

### 5.5.1 Internveiledning

Gruppen har hatt noen få møter med vår internveileder Henrik Lieng. Etter at forprosjektrapporten ble levert 23. januar, hadde vi et møte 27. januar. Her fikk gruppen tilbakemelding om at prosjektet var presentert på en god måte, men at det var en ganske stor oppgave.

Vi ble enige om at vi først og fremst skulle bruke veiledere og ressurser hos Accenture og hvert fall ha møter hver 14. dag med disse. Internveilederen sa at når det nærmet seg innlevering av rapporten skulle vi ha nye møter. I tillegg fikk vi beskjed om å ha et førsteutkast av rapporten klart en måned før innleveringsfristen. For at internveilederen

skulle holdes oppdatert brukte vi et Google-dokument for å dokumentere progresjon. Dette dokumentet ble oppdatert hver fredag. I innspurten hadde vi flere møter og leverte flere utkast. Dette ga nyttig innspill i rapportskrivingen

## 5.6 Sprintene

Til hver sprint hadde vi et dokument med brukerhistorier vi skulle jobbe med. Det ga oss en god oversikt og vi kunne lett finne fram til brukerhistoriene. Vi la så inn disse brukerhistoriene i Jira. Da kunne vi dele opp brukerhistoriene i mindre deloppgaver. Det gjorde det enklere for de andre på gruppen å følge progresjon. Dessuten ville mindre deloppgaver vise arbeidsinnsatsen på en bedre måte siden en hel brukerhistorie kan ta tid å fullføre og være uoversiktlig å følge med på, siden det er mange programvarekomponenter som henger sammen.

De første sprintene brukte vi tidsestimering med fire timer som tidsintervall. Det viste seg at de aller fleste oppgavene tok kortere tid enn forventet, og vi gikk etter hvert bort fra estimering siden vi ikke følte det tilførte noe merverdi. I tillegg skrev vi opp «use case»-diagrammer med hovedflyt og alternativ flyt. I tidlig fase av utviklingen gikk vi grundig til verks for å lage slike diagram for hver enkelt brukerhistorie. Vi erfarte etter hvert at nytten av disse diagrammene ikke kunne rettferdiggjøre tidsforbruket for de fleste av brukerhistoriene. Vi endret derfor til å bare lage «use case»-diagrammer for de mest omfattende og uoversiktlige brukerhistoriene.

### 5.6.1 Sprint 1

Brukertilhørene vi jobbet med:

- Opprette Databasen
- Som uregistrert bruker skal jeg kunne registrere meg til plattformen
- Som registrert bruker skal jeg kunne logge meg inn på plattformen

- En bruker som har glemt passord skal få et midlertidig passord på e-post
- En Innlogget bruker skal kunne opprette et nytt lag
- Deploye til Azure app service

Bortsett fra brukerhistorien «En bruker som har glemt passord skal få et midlertidig passord på e-post», lå vi godt an. E-postfunksjonalitet ble utsatt som følge av vanskeligheter. Fordi vi klarte de andre oppgavene på kort tid, fremskyndet vi en brukerhistorie «En Innlogget bruker skal kunne opprette et nytt lag» .

## 5.6.2 Sprint 2

Brukerhistoriene vi jobbet med:

- En bruker skal kunne aksessere hvilken som helst brukerprofil på plattformen
- En innlogget bruker skal kunne endre profilbilde og banner på sin egen profil
- En innlogget bruker skal kunne koble seg til eksterne spillkontoer
- En innlogget bruker skal kunne endre brukernavn, e-post og passord
- En lagleder skal kunne endre profilbilde og banner på lagsiden vedkommende administrerer
- En lagleder skal kunne invitere spillere til å være med i et lag vedkommende administrerer
- En bruker skal kunne se en oversikt over kamphistorikken til et lag på lagsiden deres.
- En bruker skal kunne se en oversikt over medlemmene i et lag på lagsiden deres

- En lagleder skal kunne gjøre andre medlemmer til lagleder

I denne sprinten var det litt større utfordringer. E-postfunksjonalitet ble ytterligere utsatt. I tillegg var funksjonalitet for opplasting av bilder og tilkoblinger til spillkontoer krevende. På daværende tidspunkt implementerte vi Steam-integrasjon som en ekstern spillkonto. Ingen på gruppen hadde erfaringer med bildeopplastning i React. Det tok litt tid å sette seg inn eksternt bibliotek for beskjæring av profilbilde og banner.

### 5.6.3 Sprint 3

I denne toukersperioden ble vi enig med produkteier om å fullføre oppgaver fra de forrige sprintene og finpusse design. I tillegg la vi til funksjonalitet for at en bruker skal kunne logge ut.

Det ble brukt mye tid på å restrukturere mappestruktur på klientsiden. I tillegg refakturerte vi nesten all serverkode slik at koden ble mer leselig. Dette førte til at alle enhetstestene måtte skrives om. Det tok også mye tid finne design og funksjonalitet for å endre brukerinnstillinger (Brukerhistorien «En innlogget bruker skal kunne endre brukernavn, e-post og passord»).

I tillegg jobbet vi med å koble til enda en ekstern spillkonto som var Discord. Det ble utsatt, som følge av utfordringer. Vi ble også enig med produkteier at det holdt å lage turneringer med kun ett format. Vi gikk for «single elimination»-turneringer eller vanlig cupspill, som de fleste kjenner det som. Dessuten endret vi appspråket fra norsk til engelsk. Vi fikk endelig e-post til å fungere som det skal.

### 5.6.4 Sprint 4

Brukerhistoriene vi jobbet med:

- En bruker skal kunne registrere en turnering
- En bruker skal kunne se på et turneringstre

I denne sprinten måtte vi bruke mye tid på design av «bracket» (turneringstre) og turneringslogikk. Det å lage et turneringstre krevde at vi satte oss inn i, og gjorde modifikasjoner på, et tredjeparts React-bibliotek.

## 5.6.5 Sprint 5

Brukerhistoriene vi jobbet med:

- En lagleder skal kunne melde seg på en turnering
- En bruker skal kunne se turneringstreet
- En bruker skal kunne se informasjonen til turneringen
- En turneringsadmin skal kunne redigere alt fra oppretting av turnering
- En bruker skal kunne se en oversikt over turneringer
- En bruker skal kunne søke etter turneringer basert på spill, land eller turneringsnavn

Jobbingen med denne sprinten gikk bra. Løsningen med et registreringsskjema på flere sider var en spennende utfordring. Dessuten måtte vi tenke nøye gjennom hvordan formatet i forskjellige faser av «single elimination» skulle være.

## 5.6.6 Sprint 6

Brukerhistoriene vi jobbet med:

- En bruker skal kunne se medlemmene i hvert lag
- Et medlem i et lag som er med i en kamp skal kunne se og bruke et chatrom
- Et medlem i et lag skal kunne rapportere kampresultatet
- Et medlem i et lag skal kunne tilkalle admin

- En admin skal kunne se alle chatrom i alle turneringer
- En turnerings-admin skal kunne se alle chatrom i sine turneringer

Vi fikk koronasmitt på et medlem innad i gruppen. Med hjemmekontor holdt de andre seg friske, men dessverre røk tidsskjema. Vi var plutselig redusert til to mann. Det var også ubeleilige at vi hadde store problemer med å få Azure app service til å fungere som det skulle. Vi var fast bestemt på å holde brukertester og ville derfor at vår Azure app service skulle fungere som forventet. I tillegg var jobben med chatrom en utfordring. Vi måtte flytte flere oppgaver til neste sprint.

## 5.6.7 Sprint 7

### Brukerhistoriene vi jobbet med:

- En systemadmin skal ha tilgang til et admin dashbord
- En bruker skal kunne chatte i kampprom
- En bruker skal kunne melde et kampresultat
- En bruker skal kunne motta varsel i navbar
- En bruker skal kunne søke etter lag, spillere og turneringer
- En turneringsadmin skal kunne endre turneringsinfo
- En bruker skal kunne melde seg på en turnering

### Andre oppgaver vi fullførte:

- Bildeopplastning ( Azure blob storage)
- Vise notifikasjonsbjelle på mindre skjermer
- E-post som fungerer på Azure app service

- SQLite til Azure SQL database
- Man kan komme til et kampron ved å klikke på en kamp i kamphistorikken
- Shadow box resize logikk (Feil som ble rettet opp)
- Spillere blir lagt til i NotifikasjonsGruppe når de besøker kampron for første gang
- Når turneringen starter blir alle admins lagt til i en notificationgruppe
- Submit result inputtene legges naturlig
- Fikse så ikke alle notifikasjoner slettes når en sletter
- Endre fra px => vh (Da skalerer innholdet til alle skjermstørrelser)

Vi greide å fullføre alt som var nødvendig med et lite unntak. Vi tok en avgjørelse på at man ikke kunne fjerne turneringsadministratorer, bare legge dem til. I tillegg strakk ikke tiden til for å «slette» lag. Vi kom frem til at om vi slettet lag ville det skape problemer i turneringstreet og kampron. Så målet var å kategorisere dem som «slettet». Dermed ville ikke turneringstreet ha plutselige hull, men som nevnt fikk vi ikke tid til å implementere dette.

### 5.6.8 Produkteliers ønsker

I løpet av statusmøtene med veiledere og produkteier, kom produkteier med ønsker. Et ønske gjaldt bytte av logo fordi markedsføringsavdelingen i Accenture ønsket en ny enklere logo. I tillegg har produkteier kommunikasjon med spillgrupper i Accenture fra andre land og han ønsket derfor at appen la til rette for at denne kunne bli brukt internasjonalt med at turneringer også hadde land som et valg. Dessuten ble språket endret til engelsk. I tillegg hadde han noen andre turneringsønsker: stokke om hvilke lag som møter hverandre i første runde og filtrering på turneringsoversikt. Med smidige utvikling har disse endringene gått fint å integrere.

## 5.7 Utfordringer

### 5.7.1 Hosting på Azure

Hosting på Azure er ikke et krav, men et nyttig verktøy for å simulere produksjon. Vi har brukt mye tid på å få det til å fungere optimalt. Vi har prøvd SQLite, Cosmos DB og Azure SQL som databaseløsning. Azure SQL har vi til slutt fått til å fungere brukbart. Vi kommer tilbake til detaljene i kapittel 6.5.5.

### 5.7.2 Discord

Discord-integrasjon var ikke høy prioritering for produkteier, men var noe vi ønsket å prøve å få til. Etter å ha møtt på problemer med discord implementering satte produkteier oss opp i et møte med en utvikler i Accenture Tyskland sin gaming-gruppe. Han hadde mye erfaring med implementering av Discord, og det var tydelig at han hadde store ambisjoner for hvordan man kunne gjøre det til en stor del av plattformen på måter vi ikke visste var mulig. At det var en med erfaring innenfor Discord som ønsket å implementere dette etter vi hadde levert bacheloren, i kombinasjon med problemene vi hadde møtt på, gjorde at vi prioriterte annen funksjonalitet. Dette var en beslutning produkteier var med å ta, da annen funksjonalitet som gjensto var mye viktigere enn Discord.

## 5.8 Nyttige triks i utviklingen

### 5.8.1 CSS-variabler

Etter hvert som prosjektet utvidet seg så merket vi at vi benyttet de samme fargekodene veldig ofte. Vi hadde også noen tilfeller hvor vi hadde som intensjon å benytte samme farge som et annet sted, men med en feiltagelse valgte en litt annen nyanse av denne fargen. Vi ønsket å finne en måte hvor vi kunne sette en

designstandard slik at utseende holdt seg konsistent for hele webapplikasjonen. CSS-variabler var løsningen på problemet vårt. Istedentfor å ha like verdier på mange forskjellige steder så definerte vi verdiene i CSS-filen som variabler og heller refererte til disse variablene. En av variablene ble benyttet så mye som 34 ganger, noe som forteller hvor nyttig dette var for oss. En bonus med denne løsningen er at man veldig enkelt kan forandre utseende på webapplikasjonen senere ved å kun endre enkeltlinjer med kode.

```
:root {  
    /* Sizing */  
    --regular-font-size: 2vh;  
    --btn-padding: 1vh 3vh;  
    --input-padding: 1vh 1.5vh;  
    /* Color */  
    --accenture-purple: #rgb(161, 0, 255);  
    --dark: #rgb(50, 50, 65);  
    --dark-opacity85: #rgba(50, 50, 65, 0.85);  
    --dark-opacity75: #rgba(50, 50, 65, 0.75);  
    --dark-opacity40: #rgba(50, 50, 65, 0.4);  
    --green: #rgb(9, 142, 50);  
    --red: #rgb(255, 0, 0);  
    --white: #rgb(255, 255, 255);  
    --grey: #rgb(166, 166, 166);  
    --invisible: #rgba(0, 0, 0, 0);  
    /* Border Radius */  
    --small-radius: 0.9vh;  
    --large-radius: 1.5vh;  
    /* Shadow */  
    --box-shadow: 0 0.5vh 1vh 0.2vh #rgba(0, 0, 0, 0.2);  
}
```

Figur 5-10 CSS-variabler

### 5.8.2 Prettier - Code formatter

En utfordring vi møtte på i starten av utviklingsprosessen var at alle gruppedellemmene hadde ulik kodestil. For å løse dette problemet tok vi bruk utvidelsen Prettier - Code formatter (Prettier) for utviklingsverktøy Visual Studio Code. Prettier brukes til å automatisk formtere koden etter bestemte regler. Alle på gruppen sørget for at å benytte de samme reglene for formatering, og innstilte utvidelsen slik at formateringen ble utført automatisk hver gang en fil ble lagret. Fordelen med denne utvidelsen var at vi både sparte tid under utviklingen av egen kode, og at det var lettere å utvide eller forbedre andre gruppedellemmers kode ettersom hele kodebasen var formatert likt.

De to bildene under er av helt identisk kode, med unntak av formateringen. Koden på Figur 5-11, som ikke er formatert, er langt raskere å skrive, mens koden på Figur 5-12, som er formatert med hjelp av Prettier, er lagt raskere å lese. Vi lastet ned utvidelsen til Visual Studio Code [her](#).

```
render() {
  const { data, errors } = this.state;
  return (
    <div className="input-form">
      <div className="form-header">
        <h1>Create team</h1>
        <hr className="purple-line line-primary"/>
      </div>
      <Input name="teamName" label="Team name" value={data.teamName} type="text" onChange={this.handleChange} error={errors.teamName} />
      <div className="purple-btn btn" onClick={this.handleSubmit}>Create team</div>
      {errors.serverErrorMessage && (
        <div className="error-message"><h3>{errors.serverErrorMessage}</h3></div>
      )}
    </div>
  );
}
```

Figur 5-11 Kode før formatering

```
render() {
  const { data, errors } = this.state;
  return (
    <div className="input-form">
      <div className="form-header">
        <h1>Create team</h1>
        <hr className="purple-line line-primary" />
      </div>
      <Input
        name="teamName"
        label="Team name"
        value={data.teamName}
        type="text"
        onChange={this.handleChange}
        error={errors.teamName}
      />
      <div className="purple-btn btn" onClick={this.handleSubmit}>
        Create team
      </div>
      {errors.serverErrorMessage && (
        <div className="error-message">
          <h3>{errors.serverErrorMessage}</h3>
        </div>
      )}
    </div>
  );
}
```

Figur 5-12 Kode etter formatering

### 5.8.3 Factories

Factories var noe vi tok i bruk etter hvert i utviklingen for å gjøre koden vår mer oversiktlig og for å ha mulighet til å gjenbruke koden flere steder uten å skrive den på nytt. En factory-klasse kan visualiseres som en fabrikk som har som eneste formål å produsere objekter eller lister med objekter av en spesifikk type.

```

public static class ProfileCardFactory
{
    public static List<ProfileCardModel> CreateList(List<Users> users)
    {
        List<ProfileCardModel> profileCards = new List<ProfileCardModel>();
        foreach (var user in users)
        {
            ProfileCardModel profileCard = new ProfileCardModel();
            profileCard.Id = user.Id.ToString();
            profileCard.ProfileImageFilename = user.ProfileImageFilename;
            profileCard.Username = user.Username;
            profileCards.Add(profileCard);
        }
        return profileCards;
    }
}

```

Figur 5-13 Factory for å produsere en liste med ProfileCardModel

### 5.8.4 Seeding av database

Det er tidkrevende å legge til brukere, lag og turneringer manuelt hver gang. Dessuten er det vanskelig å feilsøke hvis man ikke har oversikt over hele levetiden til rader i databasen. En løsning på dette i utviklingsfasen er databaseseeding. Dette gir følgende fordeler:

- Seeding av data når løsningen kompileres.
- Alle i gruppa får samme data.
- Velge om databasen blir slettet og opprettet på nytt ved kompilering

### 5.9 Refleksjon

Gjennom bacheloren har vi satt ekstremt stor pris på veilederne våre, og produkteier. Veilederne har frivillig vært med på alle møter med produkteier, selv om mange av dem ikke var obligatoriske. De har oppfordret oss på det sterkeste til å spørre om hjelp, og har vært ekstremt enkle å jobbe med. Om vi har møtt på problemer de ikke har klart å hjelpe oss med, har de ikke nølt med å bruke kontaktene sine innad i Accenture til å

få tak i noen som har kunnskapen vi trengte. Dette prosjektet hadde vært mye tøffere uten dem.

I tillegg til gode veiledere hadde vi en produkteier som hadde stor entusiasme rundt prosjektet, og var mer enn villig til å sette opp ekstra møter. Dette kunne være av grunner som at vi ville ha hans innspill på mulige designforandringer, eller at vi trengte hans mening rundt hva slags funksjonalitet som skulle prioriteres. Produkteier hadde praktisk erfaring rundt bruk av turneringsplattformer ved dataspilling. Det var derfor lett å ha produktive samtaler med han rundt funksjonalitet.

Grunnet at vi har hatt en såpass involvert produkteier, føler vi at prosjektet vi har fått en svært realistisk og praktisk innføring i hvordan det er å jobbe med en produkteier over lengre tid. Vi har gjennom prosjekter fått rikelig med erfaring med hvordan kommunikasjon mellom utvikler og produkteier skjer, og dette har vært svært lærerikt.

Som nevnt i kapittel 5.6.8, så fikk vi en oppdatering av produkteier midt i prosjektet. Denne handlet om at produktet ikke bare skulle bli brukt innad i Accenture Norge, men også i Accenture avdelinger i andre land. Et par han nevnte var Tyskland, Singapore og USA. Dette syntes vi var utrolig kult, og det gjorde så vi ble enda mer bestemt på å levere et godt produkt. Å jobbe med et prosjekt man liker er ekstremt givende, og har gjort kvaliteten på produktet bedre.

Da vi startet å planlegge prosjektet var webapplikasjonen bare ment for Accenture Norge og vi gjorde da en vurdering på at en SQLite database ville være passende med tanke på antall forventede brukere og gruppens erfaring med implementasjon av en slik database. Som en del av testingen skulle vi simulere «produksjon» og vi gjorde dette på Azure via en app service. Da ble SQLite databasen og bildemappen skrivebeskyttet. Dermed ville vi bruke en alternativ databaseløsning. Vi prøvde Cosmos DB, men fikk det ikke til og gikk derfor for SQLServer. Se kapittel 6.5.5 for flere detaljer. Migreringen til SQLServer var lagt mer tidkrevende enn forventet, og var noe som hold progresjonen vår litt tilbake.

Siden vi har investert så mye tid på hosting på Azure og lenge hadde en løsning som ikke fungerte burde vi bedt om mer hjelp. Vi visste at Accenture har tilgjengelige ansatte med god kompetanse på området.

Da en av gruppens medlemmer ble smittet av korona har vi tidligere nevnt at tidsskjema røk. Vi ser i ettertid at vi burde satt opp en ny prioritering av arbeidsoppgaver med redusert kapasitet. Vi var såpass inni det vi holdt på med at vi fortsatte med den samme arbeidsfordelingen.

Som tidligere nevnt var React noe gruppen ikke hadde veldig mye erfaring med før prosjektstart. Derfor valgte vi å starte med de antatt enkleste oppgavene for å bygge opp erfaring og kunnskap før vi startet på det vi så for oss var mer krevende. Når vi ser tilbake på utviklingen, så føltes det ikke ut som oppgaven ble vanskeligere utover i prosjektet, noe vi tror skyldes at vanskelighetsgraden på oppgavene og ferdighetsnivået innad i gruppen økte i tandem.

Vi merket at det tok veldig mye tid å gjennomføre alle delene av Scrum til punkt og prikke. Ettersom vi hadde et såpass tett samarbeid med hadde videomøter langt hyppigere enn de som var fast planlagt hver andre dag, så vurderte vi faste møter for tidsestimering og retrospekt som overflødig og unødvendig tidkrevende.

# 6 Produktdokumentasjon

## 6.1 Forord

Produktdokumentasjonen er ment for både personer med og uten teknisk kunnskap. For å få fullt utbytte av kapittel 0, programmets oppbygning og virkemåte, kreves det i større grad teknisk kunnskap.

## 6.2 Kort beskrivelse av produktet

Accenture e-sport er en webapplikasjon utviklet for Accenture Norge sin spillgruppe. Det er en turneringsplattform som lar spillgruppen unngå alt det manuelle arbeidet i programmer som Microsoft Excel og Microsoft Teams, for å holde en turnering. Det er enkelt å registrere en bruker. Da kan man bli med på et lag enten ved å bli invitert eller opprette et lag selv. Gjennom en utlisting av alle turneringer på siden kan man så melde seg på en av disse med et av lagene man er leder for. Om man selv ønsker å opprette en turnering kan også hvem som helst gjøre dette enkelt, og legge til bekjente som administratorer for å lettere driftet turneringen.

## 6.3 Hovedfunksjonalitet

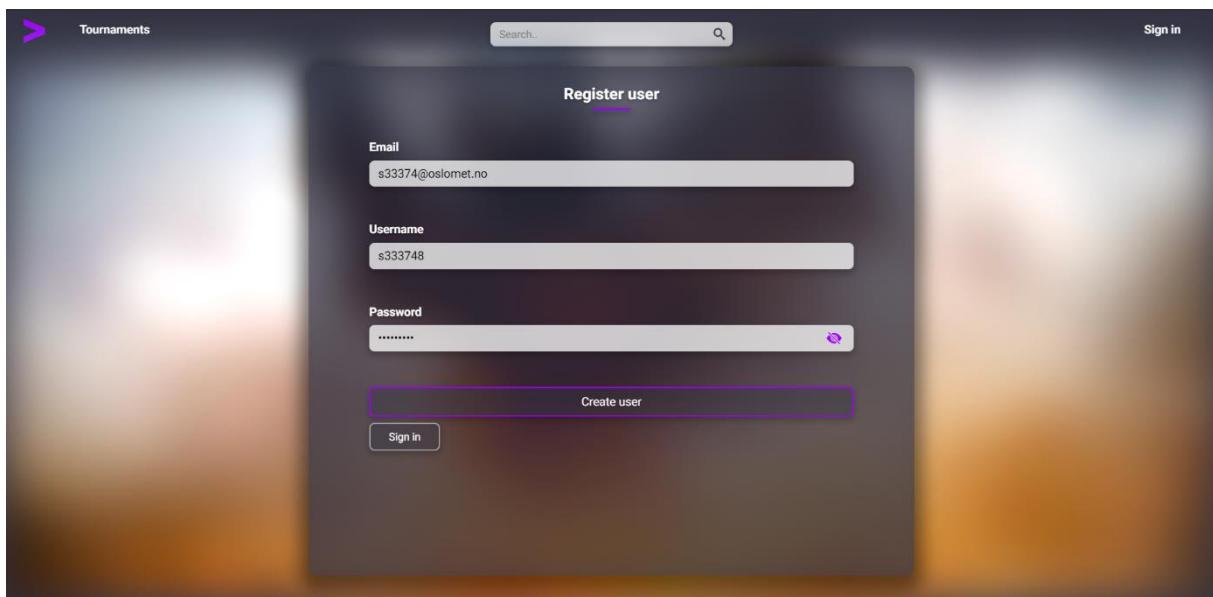
I denne delen av kapittelet skal vi beskrive hovedfunksjonaliteten og få frem samsvar mellom kravspesifikasjonen og produktet.

### 6.3.1 Registrering og innlogging

#### 6.3.1.1 Registrering

Som en uregistrert bruker er det mulig å registrere seg på «Sign up»-siden. Da trykker brukeren på en «sign up»-knapp, og blir tatt til et registreringsskjema, som vist på Figur 6-1. Etter at man har registrert seg som en ny bruker blir man automatisk logget inn, og

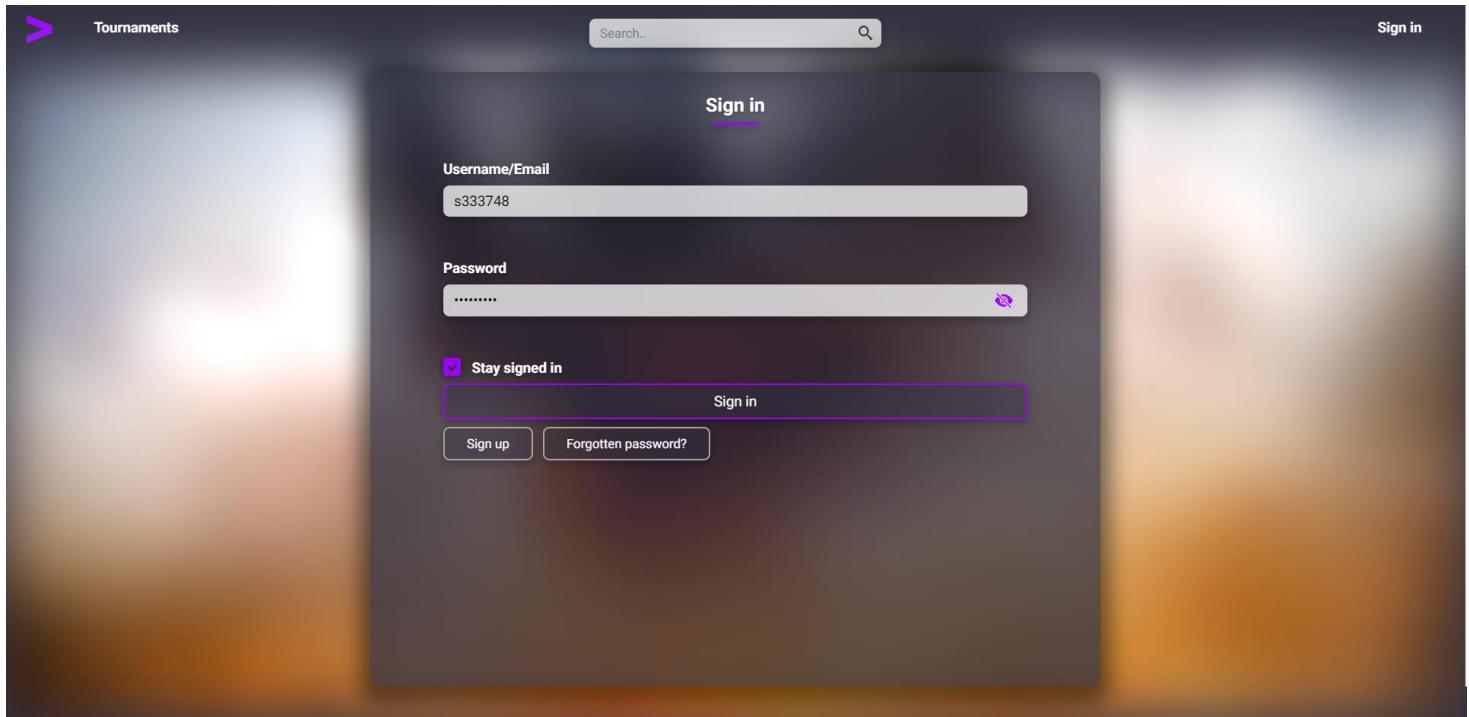
tatt til sin egen bruker profil. Passordet man skriver inn i registreringsskjemaet blir kryptert gjennom en krypteringsalgoritme, og blir plassert i databasen sammen med resten av brukerne. Systemet lagrer aldri passordet, og har ikke mulighet til å finne ut av hva det er. Se kapittel 6.4.6.4 for mer informasjon. Passordet må også inneholde minst åtte tegn, hvor to av disse er et tall og en stor bokstav. Om man ønsker å se passordet sitt før man prøver å logge inn kan man trykke på øyet-ikonet på innsiden av passord feltet. Da blir passordet gjort om til klartekst.



Figur 6-1 Registreringsskjema fylt inn med brukerinformasjon

### 6.3.1.2 Innlogging

Som en bruker som allerede har en profil skal det være mulig å logge seg inn. Dette gjøres gjennom et innloggingsskjema som vist i Figur 6-2. Her kan man velge om man ønsker å fylle inn brukernavn eller e-post, i tillegg til passord. Dette passordet blir kryptert gjennom samme krypteringsalgoritme som under registrering, og sammenlignet med passordet som ligger kryptert i databasen. Om disse matcher blir man logget inn. Man kan også huke av avkryssingsruten hvor det står «Stay signed in» for å slippe å logge inn ved neste besøk av nettsiden. (Se kapittel 6.4.6.5)



Figur 6-2 Innloggingsskjema fylt inn med brukerinformasjon

### 6.3.1.3 Gjenopprettning av glemt passord

Som en bruker som har registrert seg, men ikke husker passordet sitt er det mulig å få tilsendt en e-post med aktiveringskode for å bekrefte identiteten sin. Denne prosessen vises i Figur 6-3. Etter å ha skrevet inn e-posten sin, og trykket på send-knappen blir det sendt en e-post med aktiveringskode. Da dukker det opp et felt hvor man kan bekrefte aktiveringskoden man har fått tilsendt. Når brukeren har skrevet inn aktiveringskoden blir den sendt til en ny side for å velge et nytt passord. Det forutsettes at aktiveringskoden er den samme som i e-posten. Denne aktiveringskoden lagres i databasen for å kunne sjekke dette. På «oppdatere passord»-siden må brukeren velge et passord som tilfredsstiller våre passordregler. Etter dette må man skrive inn passordet på nytt i et identisk felt, for å bekrefte det. Når passordet oppdateres blir det kryptert på samme måte som ved registrering. (Se kapittel 6.4.6.4)

**Forgot password**

Please enter the registered email address

**Send email with activation code**



**Forgot password**

Please enter the registered email address

**Send email with activation code**

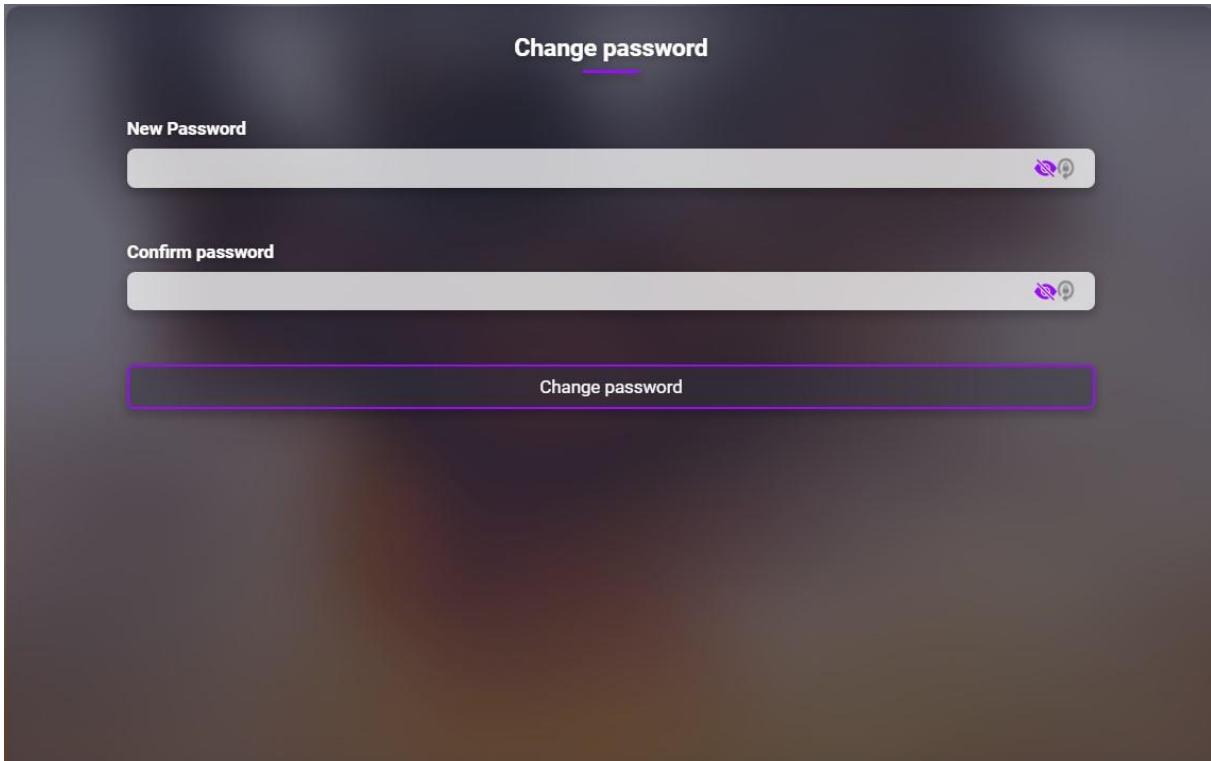
Have you not received any email? Send email again.

**Activation code**

**Please enter 6-digit activation code**

**Go to change password**

Figur 6-3 Send aktiveringskode per e-post



Figur 6-4 Oppdatere passord

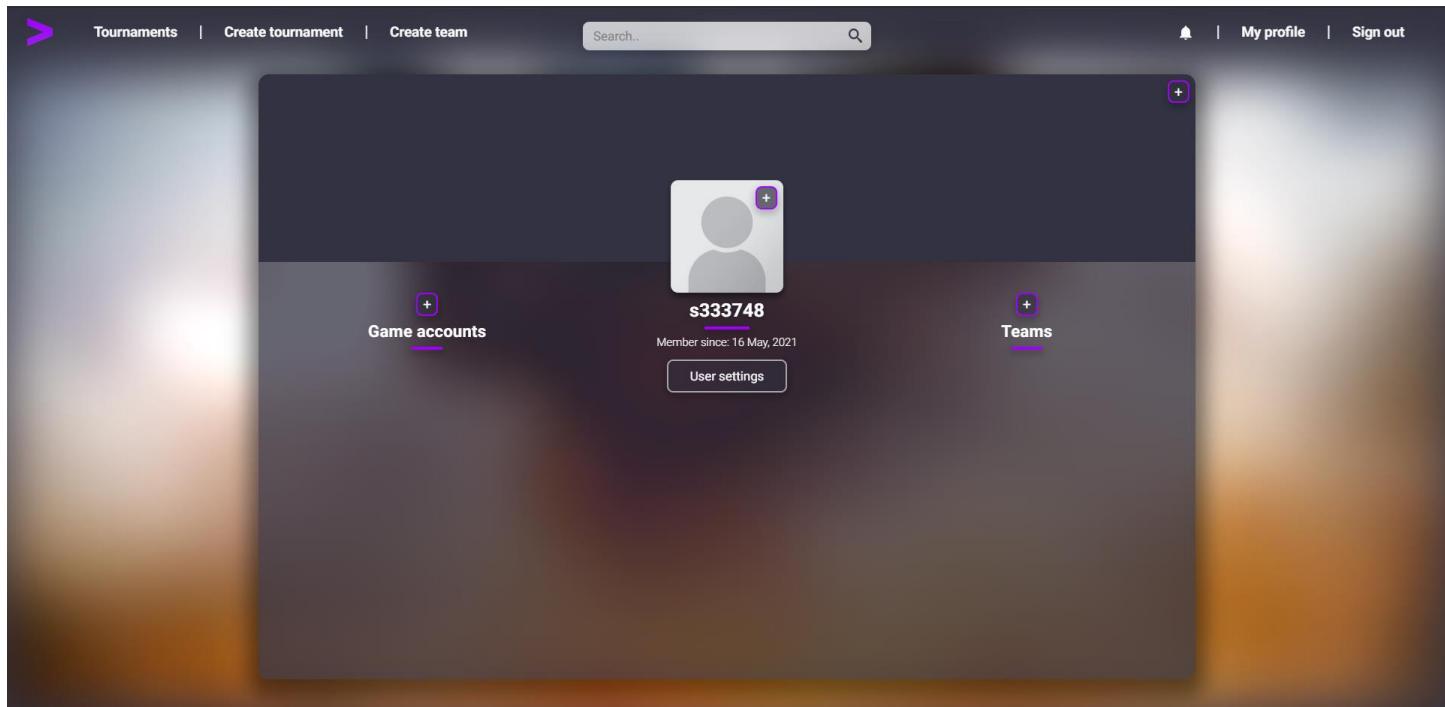
#### 6.3.1.4 Refleksjon over kravspesifikasjonen

Den originale kravspesifikasjonen sier at «Nye brukere skal kunne registrere seg og logge inn på plattformen. Brukerstøtte for gjenopprettning av passord skal også implementeres.» I tillegg inkludere den de ikke-funksjonell kravene: «Passord skal lagres kryptert i databasen ved hjelp av en hashingalgoritme». «Passordene skal være minst åtte tegn med minst et tall og en stor bokstav». Disse kravene er oppfylt.

#### 6.3.2 Brukerprofil

Brukerprofilen inneholder all informasjon om en enkeltspiller og en oversikt over alle lagene man spiller for. Det er mulig å knytte opp Steamprofilen sin til brukerprofilen. Steam er en av verdens største spillplattformer. Det er også mulig å bytte profilbilde og banner. Dette blir gjort gjennom de lilla knappene med et plussstegn inni, og den

store knappen hvor det står «User settings» som vist på Figur 6-5. Disse blir brukt til å redigere brukeren, og kommer videre til å bli referert til som redigeringsknapper. Redigeringsknappene er bare synlig for eieren av brukerprofilen.

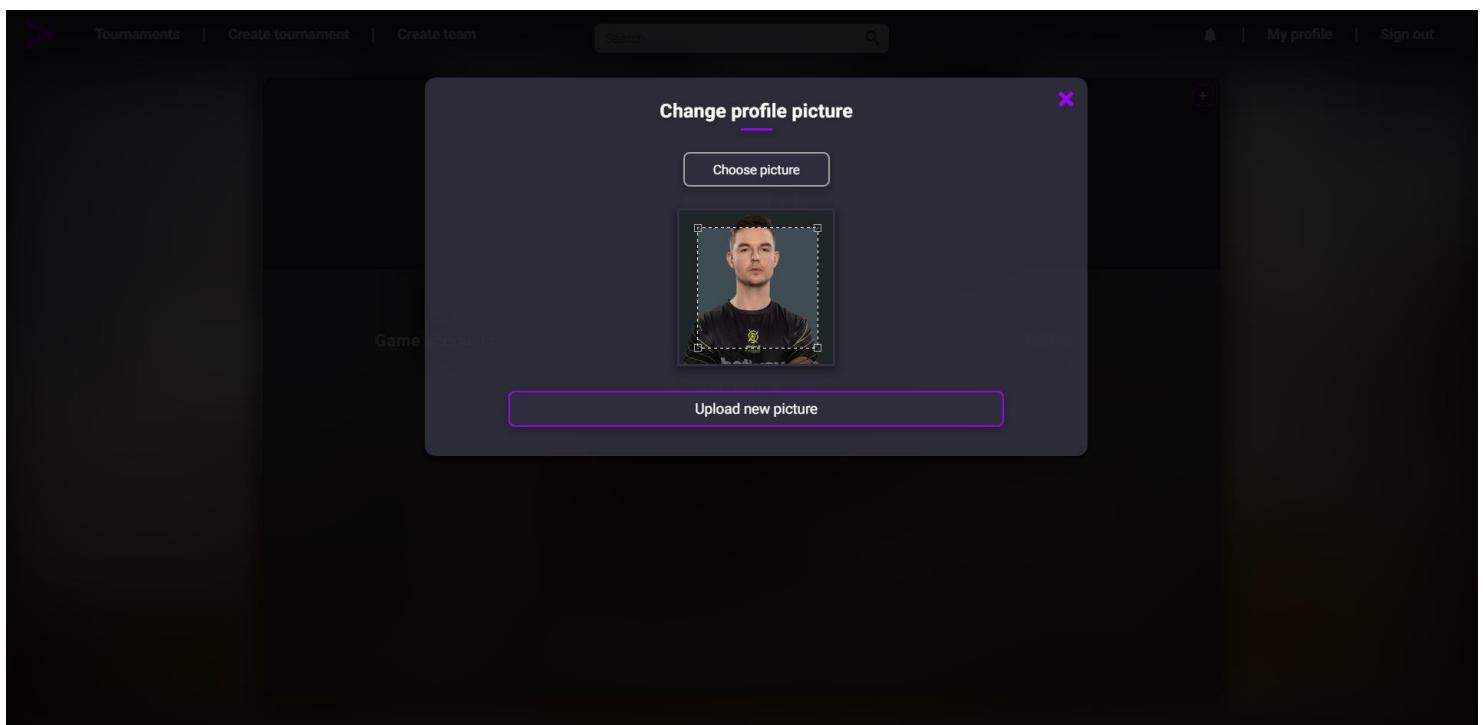


Figur 6-5 Uredigert brukerprofil

### 6.3.2.1 Profilbilde og banner

For å bytte profilbilde trykker man på redigeringsknappen på innsiden av profilbildet. Da åpnes det et modal-vindu hvor man kan velge seg et profilbilde fra egen pc gjennom den innebygde filutforskeren, og tilpasse det ved hjelp av et beskjæringsverktøy. Beskjæringsverktøyet vises i Figur 6-6. Man bytter bannerbilde på tilsvarende måte.

Mulighet til å bytte profilbilde og banner er inkludert i den originale kravspesifikasjonen. Dette kravet er innfridd.

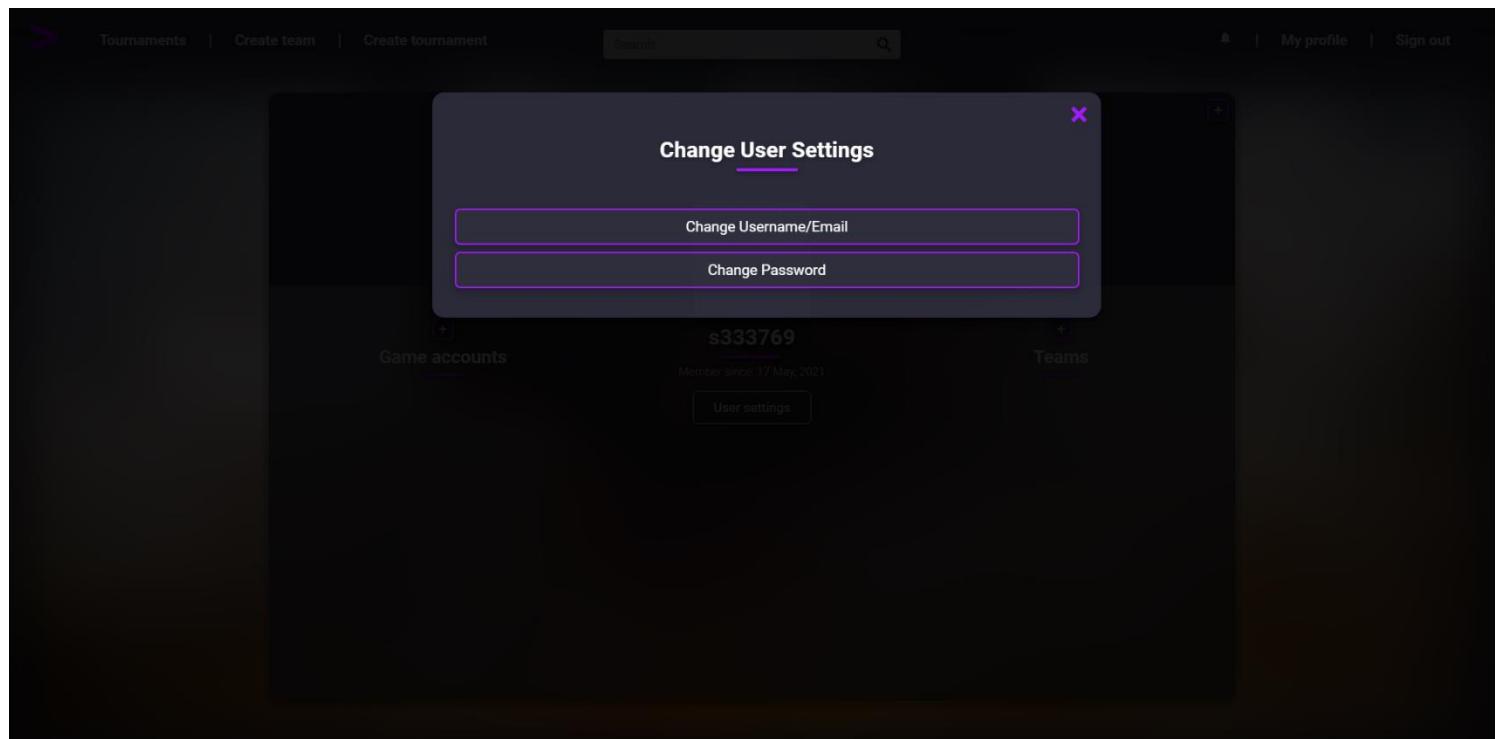


Figur 6-6 Beskjæringsverktøy for profilbilde og banner

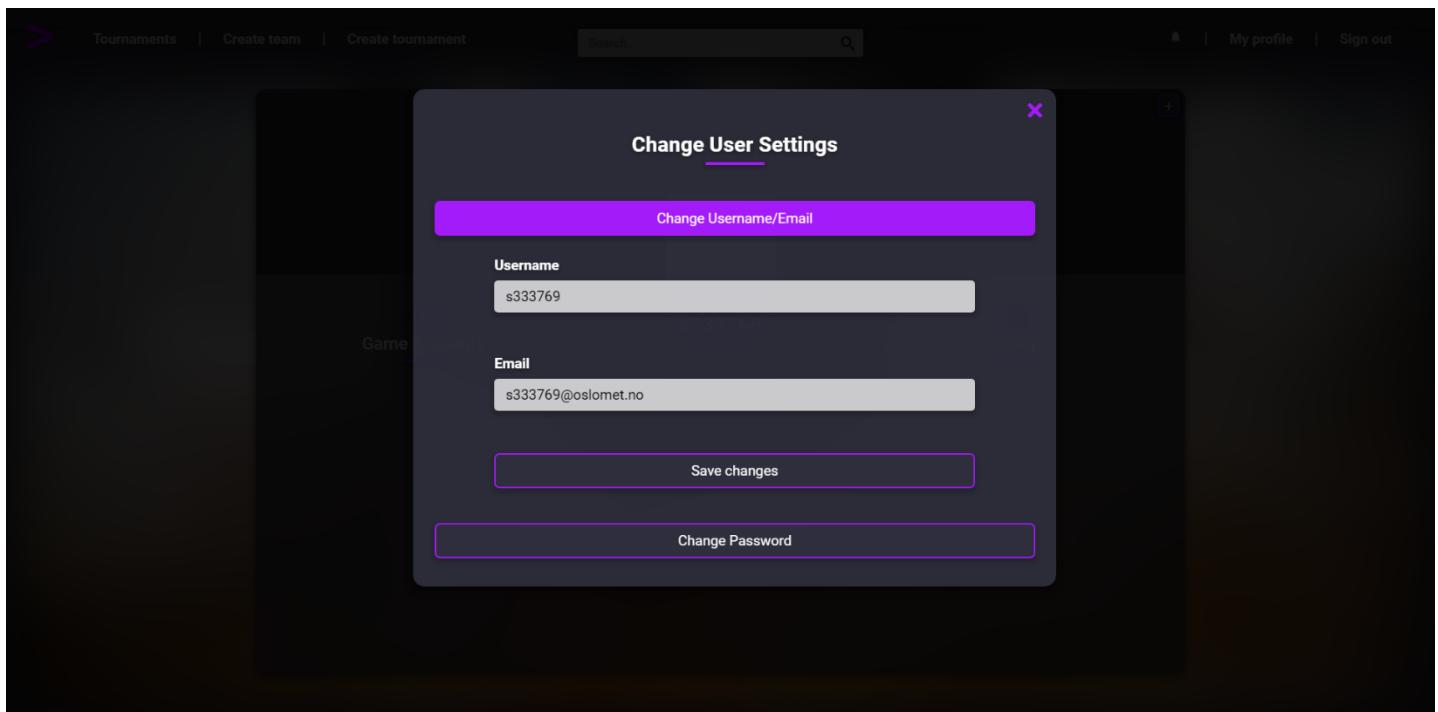
### 6.3.2.2 Oppdatering av brukerinformasjon

Som en innlogget bruker er det mulig å bytte personlig brukerinformasjon som brukernavn, e-post og passord. Dette er mulig da vi har valgt å identifisere en bruker gjennom en unik id, framfor å benytte brukernavn eller e-post. For å forandre brukerinformasjon kan en bruker trykke på redigeringsknappen på brukerprofilen, hvor det står «User settings». Da åpnes det et modal-vindu hvor brukeren har mulighet til å forandre brukerinformasjonen sin, som vist i Figur 6-7, Figur 6-8 og Figur 6-9. Her ligger brukernavn og e-post allerede fylt inn. Når en bruker skal forandre passord må han/hun først fylle inn sitt nåværende passord, samt sitt nye passord to ganger. På denne måten hindres en inntrenger i å endre passordet uten å vite brukerens passord. Det blir også mindre sannsynlig at brukeren gjør en tastefeil og må gå gjennom prosessen for gjenoppretting av passord.

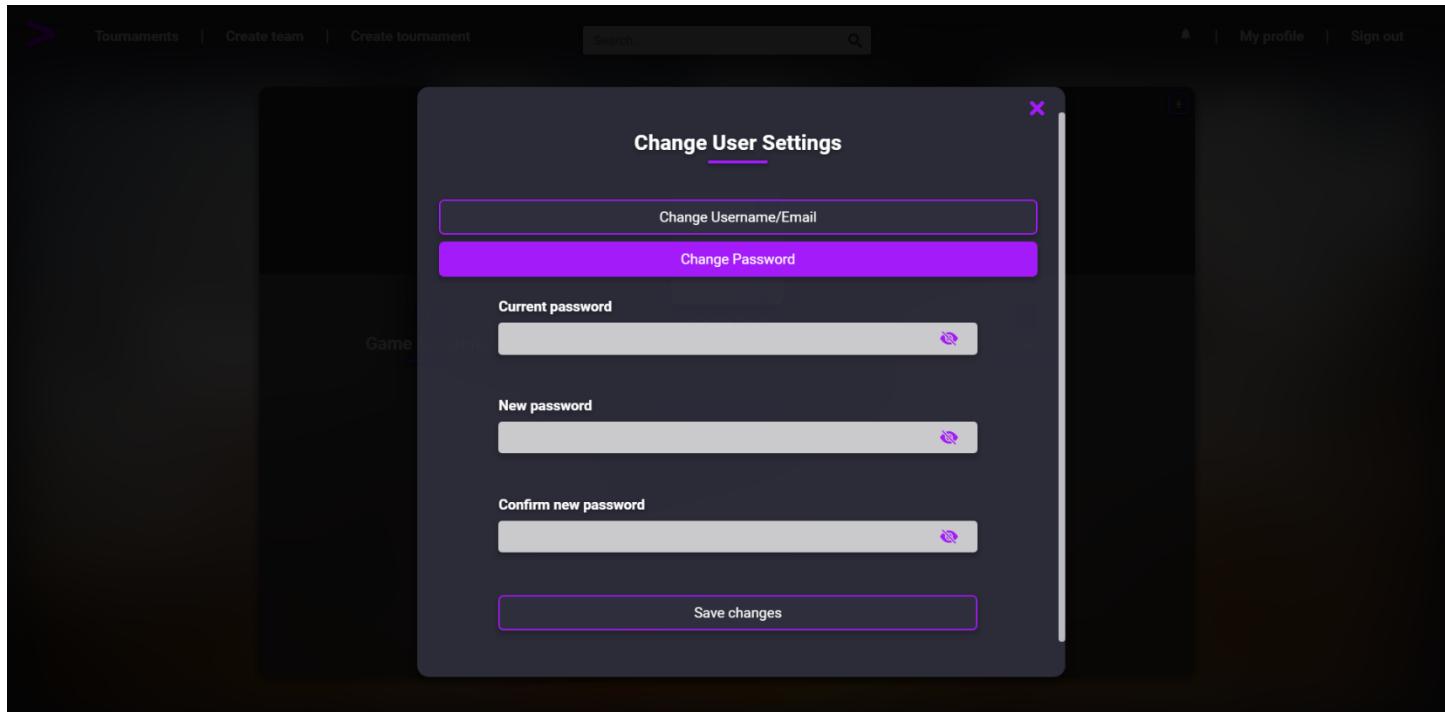
Oppdatering av brukerinformasjon ble ikke ført opp i kravspesifikasjonen, men er standard funksjonalitet for brukerprofiler. Produkteier mente også dette var viktig å få med da det ble tatt opp i sprint-planlegging.



Figur 6-7 Lukket modal-vindu for oppdatering av brukerinformasjon



Figur 6-9 Modal-vindu hvor bruker ønsker å oppdatere brukernavn eller epost

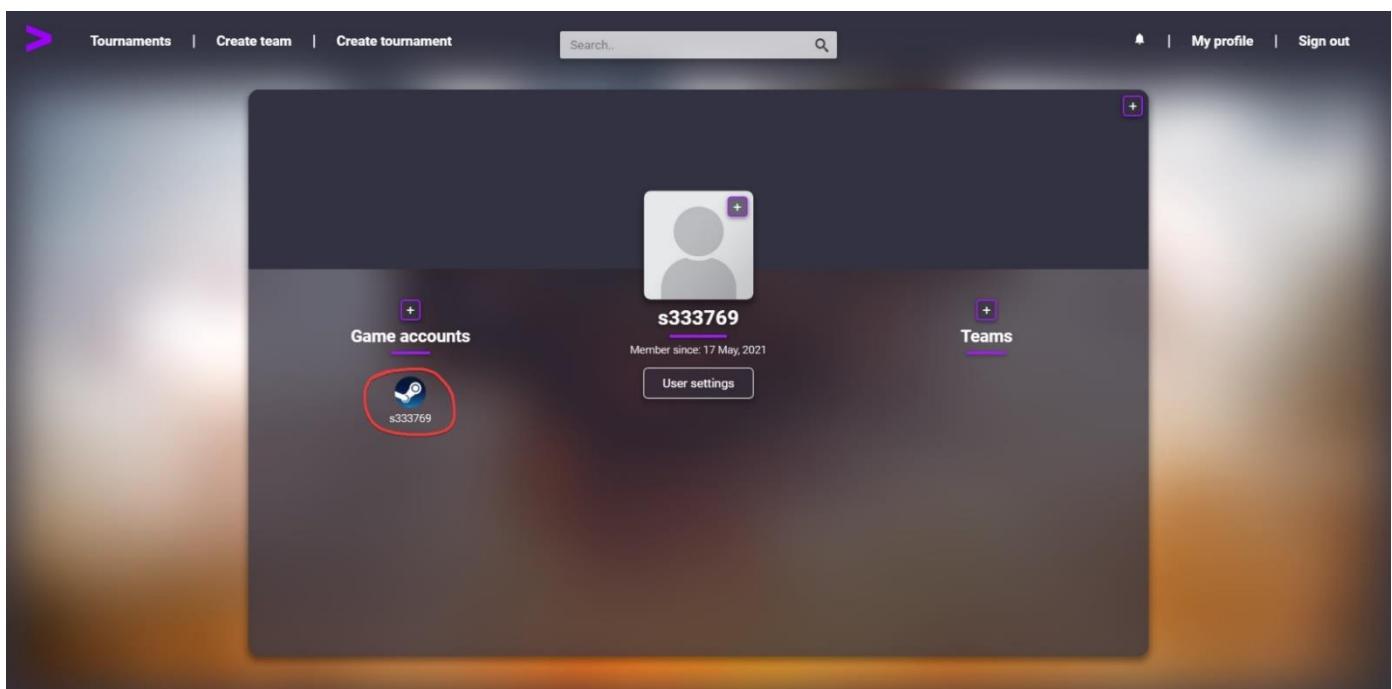


Figur 6-8 Modal-vindu hvor bruker ønsker å oppdatere passord

### 6.3.2.3 Tilkobling til eksterne spill kontoer

Som nevnt tidligere så har en bruker muligheten til å koble opp Steam til brukerprofilen sin. Når man spiller dataspill, er det på mange forskjellige plattformer. Vår plattform gjør det mulig å drive og spille turneringer. Her kan man møte nye mennesker, og knytte nye bånd. Om man møter nye mennesker via plattformen vår, ønsker vi å gjøre det lettere for dem å spille med hverandre. Da kan man gå inn på hverandres brukerprofil, og klikke på Steam-ikonet under «Game accounts», som vist i Figur 6-10. Da blir man tatt til Steam-profilen til denne brukeren, og kan legge dem til der.

En annen viktig funksjon å kunne legge til Steam-profilen sin har, er at man beviser at man er den man utgir seg for å være. Dette kan man vite siden spilleren må gjennom Steam sin autentiseringsflyt. Dette blir beskrevet i nærmere detalj i neste avsnitt. Om en kjent proffspiller skulle lagd en profil på plattformen vår, og lagt til Steam-kontoen sin ville dette altså verifisert at denne spilleren er den han utgir seg for å være.

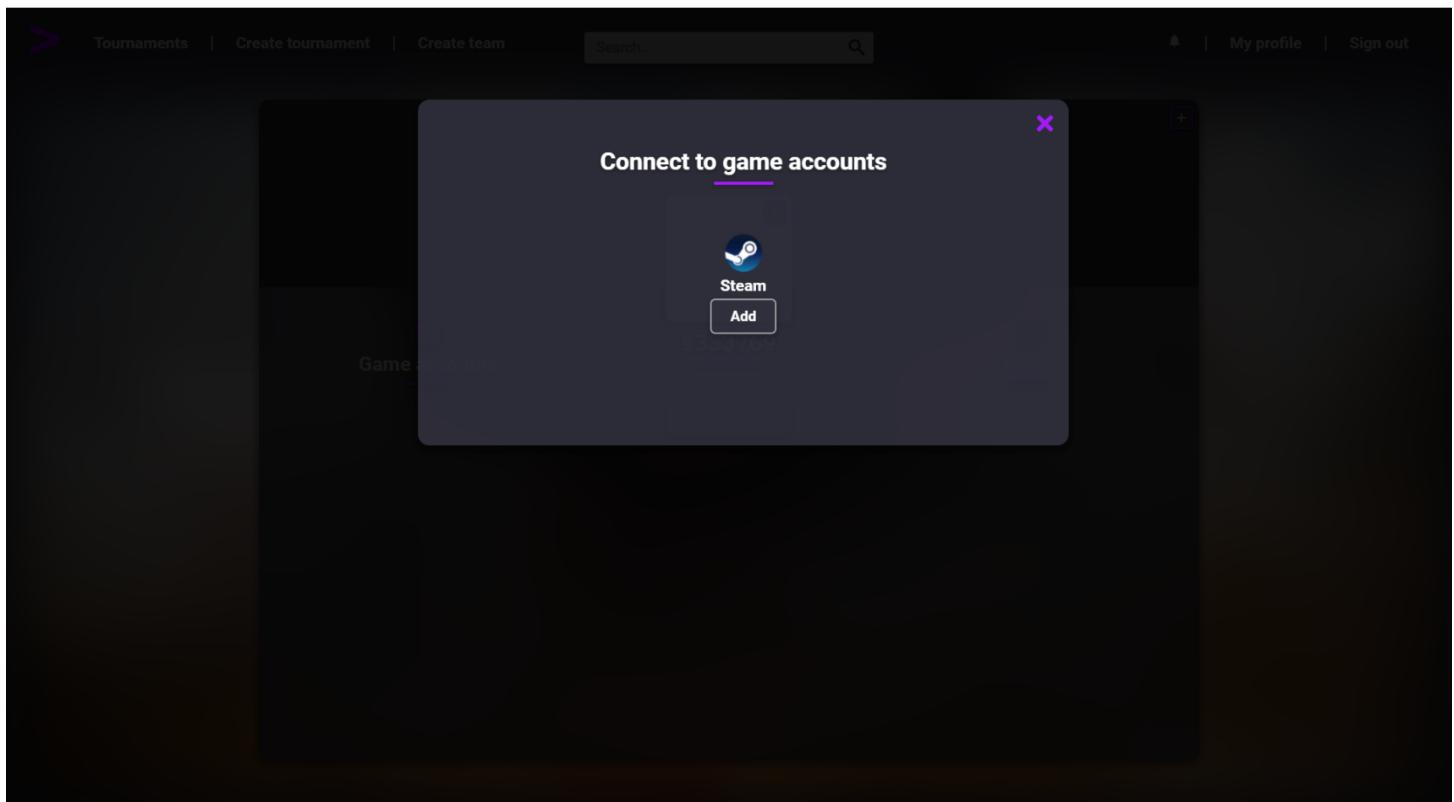


Figur 6-10 Brukerprofil koblet opp mot Steam-konto

I forrige avsnitt ble det nevnt at en bruker kan legge til Steam på sin brukerprofil gjennom steam sin autentiseringsflyt. For å komme seg til denne autentiseringsflyten må en bruker trykke på redigeringsknappen over «Game accounts» overskriften. Da åpnes et modal-vindu hvor man har mulighet til å legge til Steam, som vist i Figur 6-12. Når Accenture tar over prosjektet, vil det bli lagt til flere spillkontoer her. Mer om dette i neste avsnitt. Når man trykker på «Add» under steam-ikonet i modal-vinduet blir man sendt til Steam sin autentiseringsflyt, som vist i Figur 6-11. Etter å ha autentisert seg med Steam brukeren sin blir man tilbakesendt til turneringsplattformen, sammen med en unik identifikator kalt en SteamID. Alle Steam-profiler har en unik SteamID, og gjennom denne kunne vi gjøre steam-ikonet til en hyperlink til brukerens Steam-profil.

Kravspesifikasjonen sier at det skal være mulig å knytte opp andre spillprofiler til brukerprofilen. Vi har prøvd å implementere to spillprofiler i Discord og Steam, men møtte på litt problemer med Discord. Grunnen til at vi valgte å bortprioritere Discord er dokumentert under «Utfordringer» i Prosessdokumentasjonen.

Figur 6-12 og Figur 6-11 viser modal-vinduet for spillkontoer, og steam sin autentiseringsflyt.



Figur 6-12 Modal-vindu med mulighet til å legge til eksterne spillprofiler

Sign into accentureesportportal.azurewebsites.net using your Steam account

 Sign in through STEAM

**i** Note that accentureesportportal.azurewebsites.net is not affiliated with Steam or Valve

Steam username

Password

**Sign In**

By signing into accentureesportportal.azurewebsites.net through Steam:

- Your Steam login credentials will not be shared.
- A unique numeric identifier will be shared with **accentureesportportal.azurewebsites.net**. Through this, accentureesportportal.azurewebsites.net will be able to identify your Steam community profile and access information about your Steam account according to your **Profile Privacy Settings**.
- Any information on your **Steam Profile** page that is set to be publicly viewable may be accessed by accentureesportportal.azurewebsites.net.

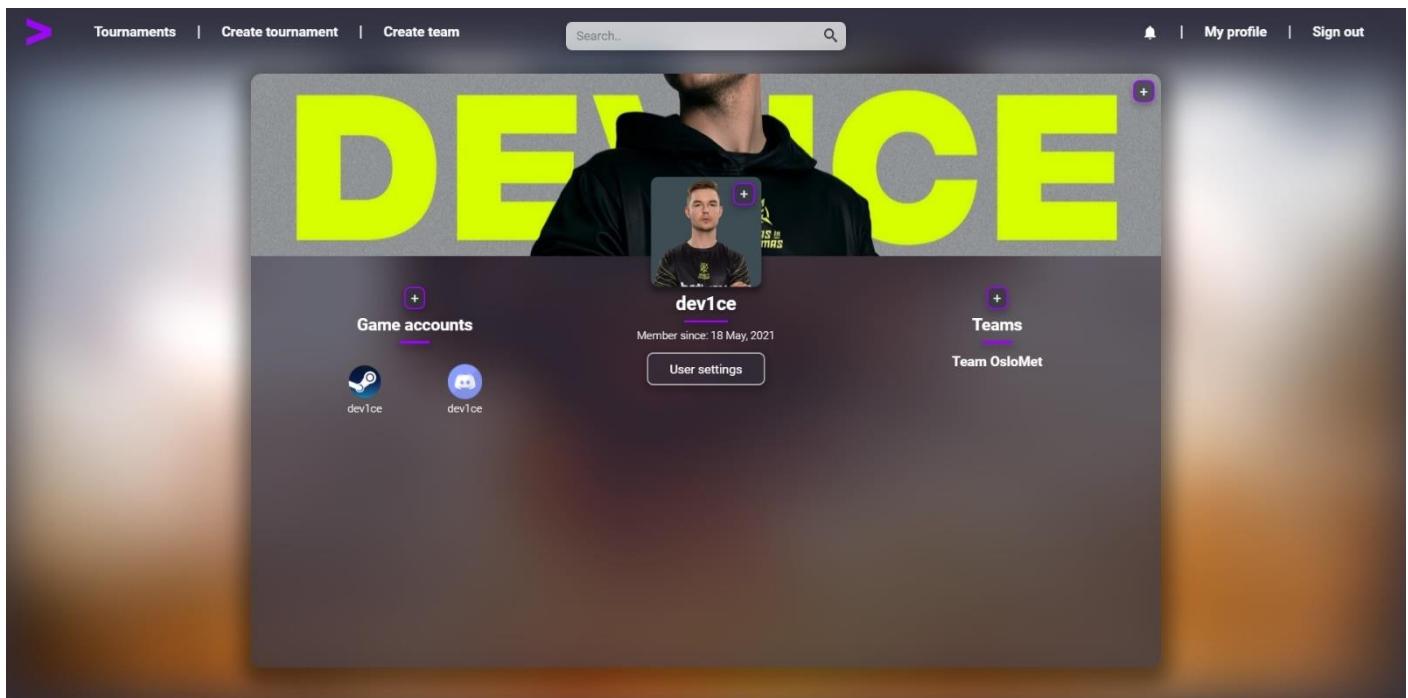
By clicking "Sign In" you agree to this data being shared.

Don't have a Steam account? You can [create an account](#) for free.

Figur 6-11 Steam autentiseringsflyt

#### 6.3.2.4 Fullstendig brukerprofil

Figur 6-13 viser en fullstendig brukerprofil hvor alle redigeringsknappene er tatt i bruk.

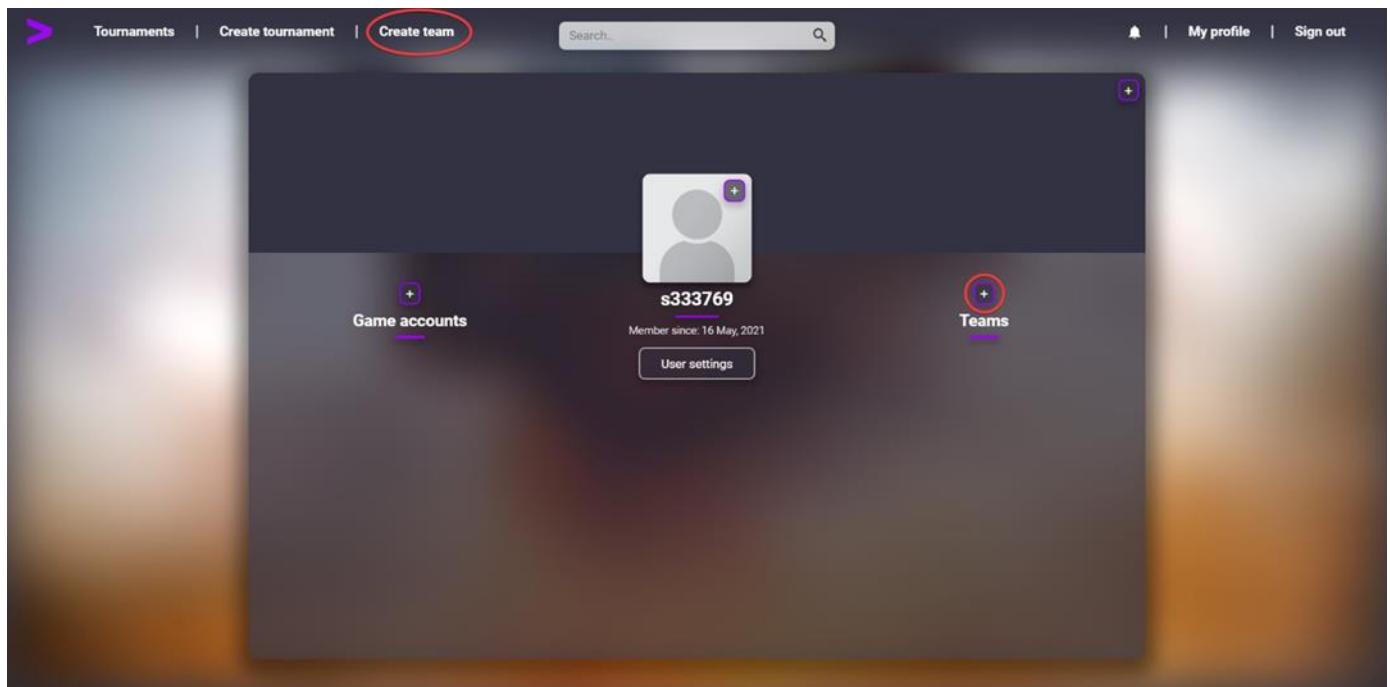


Figur 6-13 Brukerprofil hvor alle redigeringsknappen er tatt i bruk

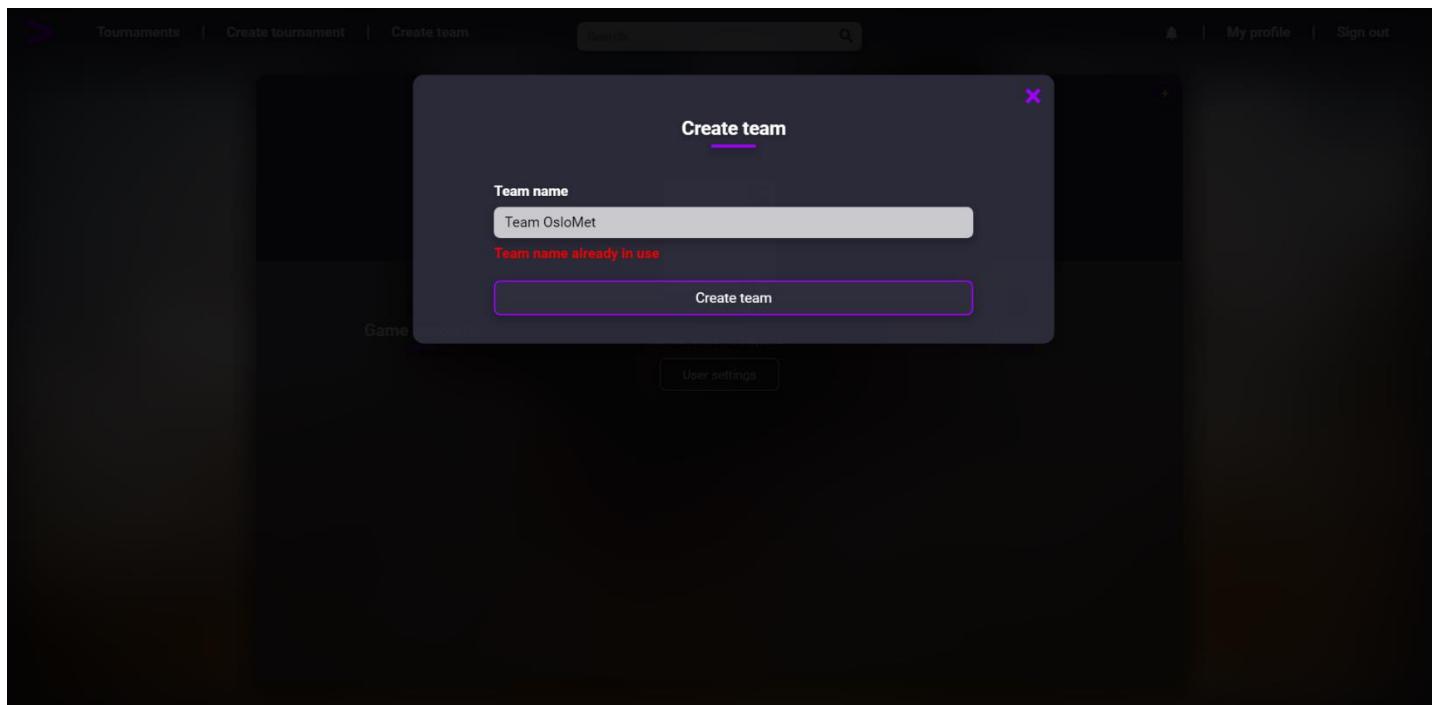
#### 6.3.3 Opprettning av lag

Som en bruker med vanlige rettigheter på turneringsplattormen, skal man kunne opprette et eget lag. Dette gjøres enten fra brukerprofilen ved å trykke på redigeringsknappen over «Teams» overskriften, eller ved å trykke på «Create team» i navbaren. Dette vises i Figur 6-14. Da forandres innholdet i vinduet til et input-felt hvor man kan skrive inn navnet til laget man ønsker å opprette. Når man har trykket på knappen for å opprette laget får man enten en feilmelding som sier at lagnavnet allerede er i bruk, eller så opprettes laget. Dette er vist i Figur 6-16 og Figur 6-15. Etter dette blir man tatt direkte til den nye lagprofilen.

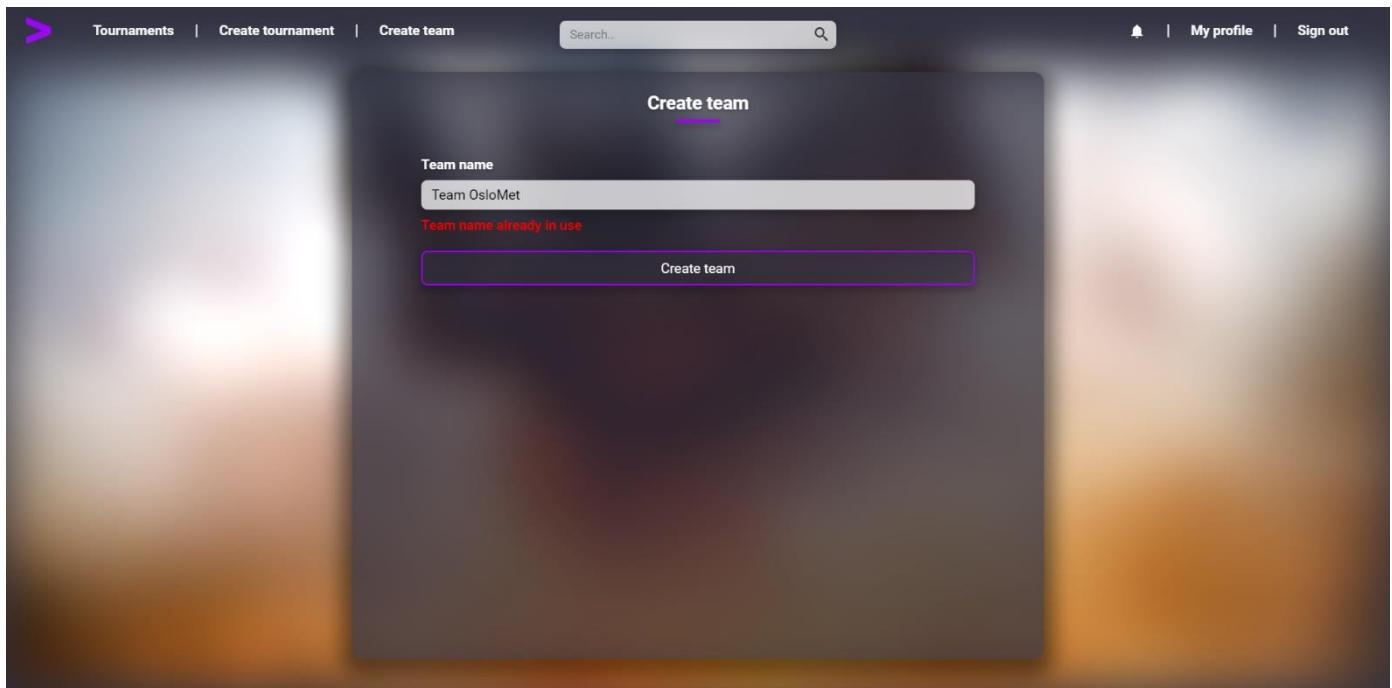
Mulighet for å opprette et lag er inkludert i den originale kravspesifikasjonen. Dette kravet er innfridd



Figur 6-14 Bruker kan opprette lag på to måter



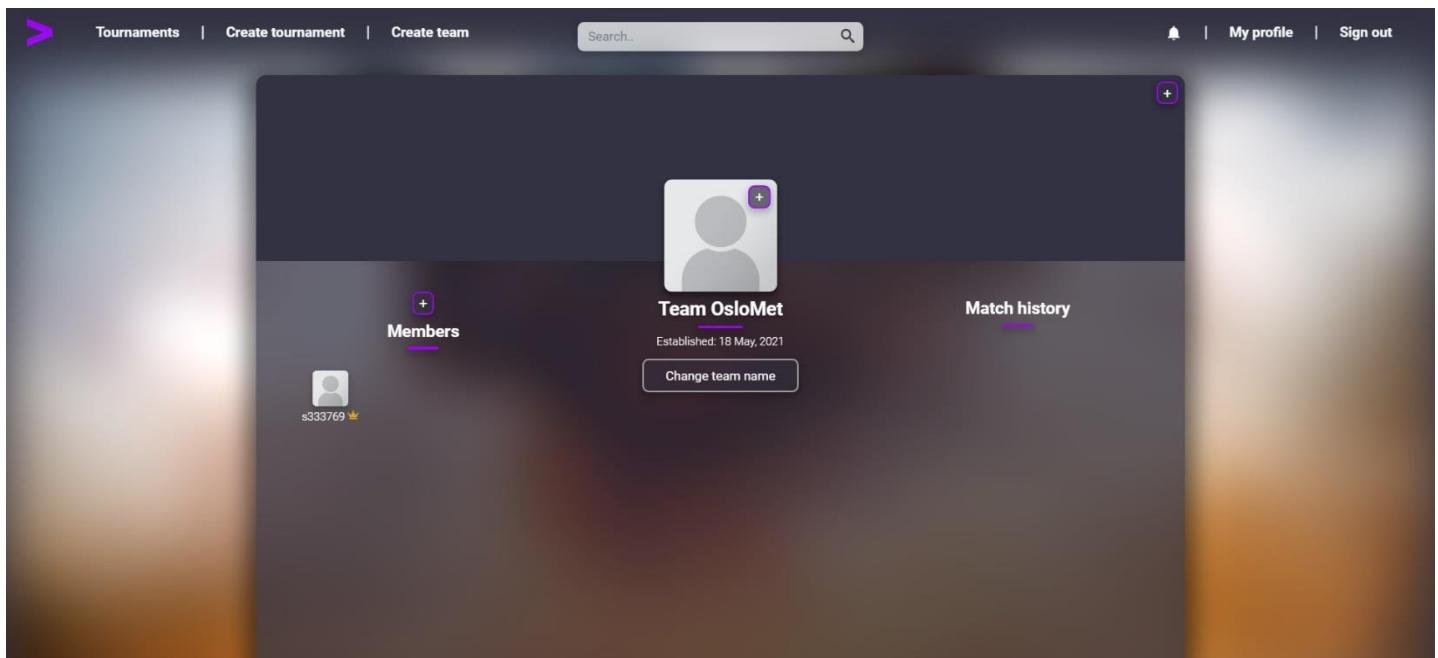
Figur 6-15 bruker prøver å opprette lag med opptatt navn gjennom brukerprofil



Figur 6-16 Bruker prøver å opprette lag med opptatt navn gjennom navbarknappen

### 6.3.4 Lagprofil

Lagprofilen har gjenbrukt store deler av designet fra brukerprofilen. Komponentene som er brukt for profilbilde og banner er gjenbrukt, samt beskjæringsverktøyet som interagerer med disse. Strukturen med to kolonner, en på høyre side av profilen, og en på venstre side, går igjen. En uredigert lagprofil vises i Figur 6-17.



Figur 6-17 Uredigert lagprofil

#### 6.3.4.1 Medlemmer

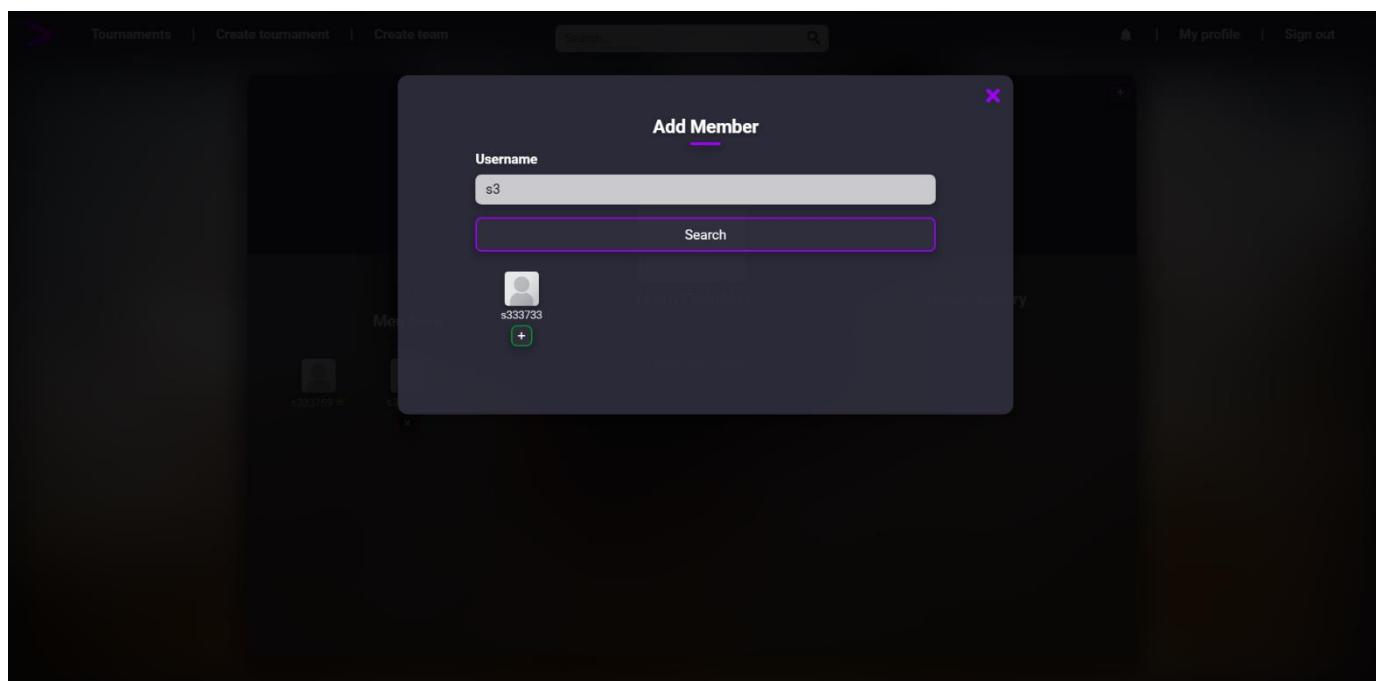
Et lag inneholder brukere med to typer roller. Lagledere, og medlemmer. En lagleder har rettigheter som å forandre på informasjonen til laget gjennom redigeringsknappene, og muligheten til å melde et lag på en turnering. I tillegg kan en lagleder invitere andre brukere til laget, og fjerne dem. Redigering av lagnavn, og bilder gjøres på samme måte som på brukerprofilen.

Et medlem har ingen rettigheter på lagprofilen, men har tilgang til alle kamprom laget er med i, og får varsler som rettes mot alle lagets medlemmer. Se kapittel 6.3.10 og kapittel 0 for mer informasjon. I tillegg til lagledere og medlemmer blir alle brukere som er inviterte til laget vist på lagsiden, men disse er bare synlige for lagleder.

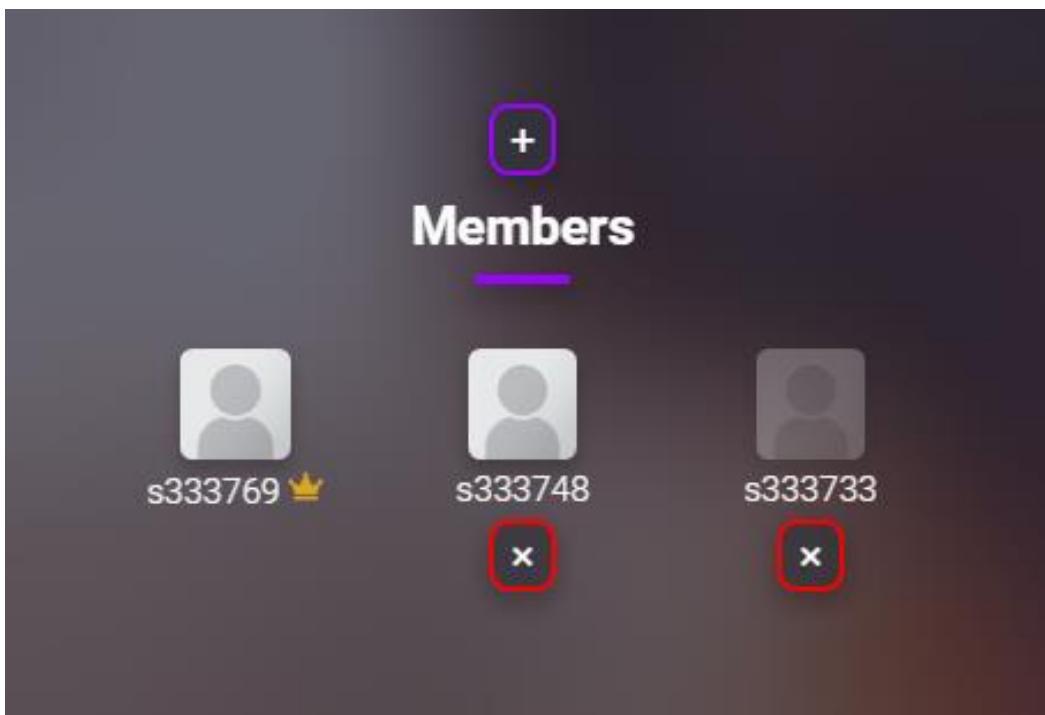
#### 6.3.4.2 Invitering og fjerning av medlemmer

Som gjennomgått tidligere er det bare lagleder som har muligheten til å invitere og fjerne spillere fra lagoppstillingen. For å invitere en spiller til laget må laglederen trykke på redigeringsknappen over «Members» overskriften i venstre kolonne av lagprofilen. Da åpnes det et modal-vindu som brukes til å søke etter spillere, som vist i Figur 6-18. Ved å trykke på den grønne knappen under profilen blir brukeren invitert, og ved å trykke på profilen blir man tatt til brukerens profilside. Dette er nyttig om det er flere brukere med liknende navn, og man er usikker på hvilken av brukerne som er personen man ønsker å invitere.

Figur 6-19 viser oversikten over alle brukere på et lag. Laglederen har en krone for å symbolisere rollen, og en invitert spiller er litt gjennomsiktig for å tydeliggjøre, for lagleder, at denne spilleren er invitert. For å fjerne en spiller trykker lagleder på den røde knappen under spillerprofilen. Da må den bekrefte handlingen gjennom en modal. Dette er for å unngå å fjerne spillere ved feiltagelse.



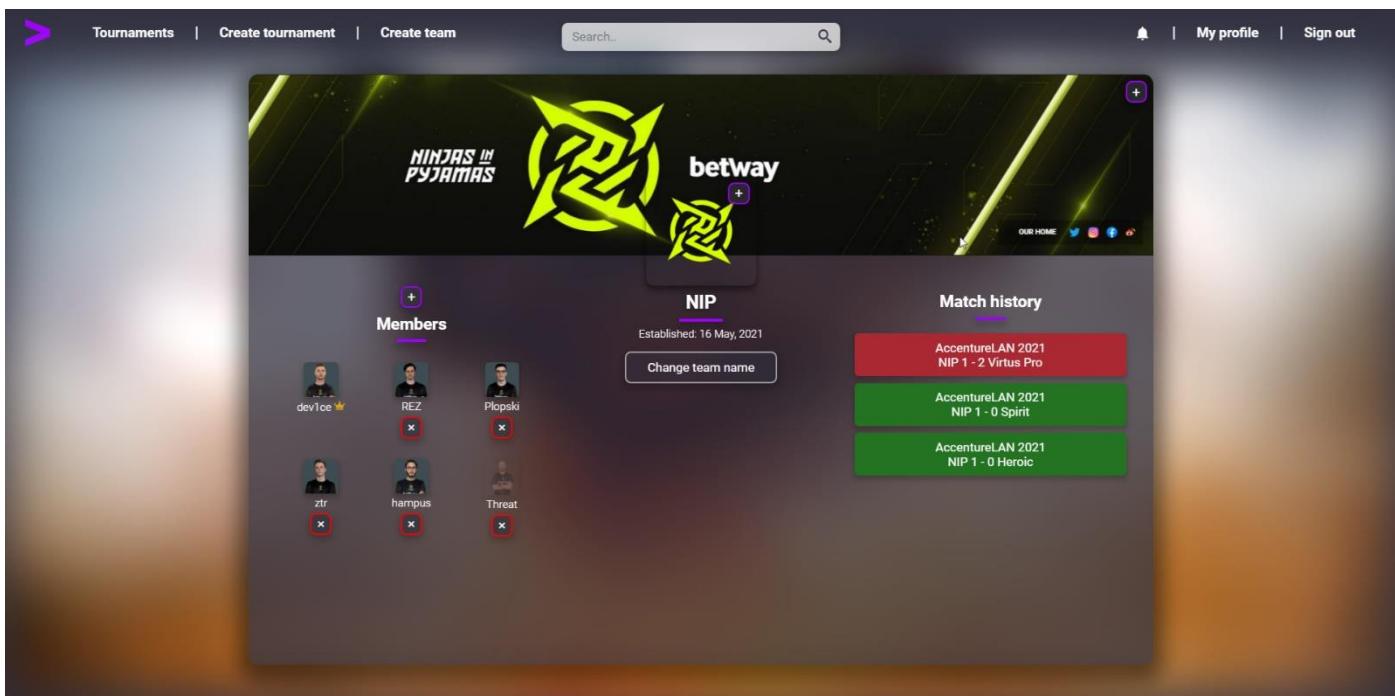
Figur 6-18 Modal-vindu for å søke opp spiller. Bare s333733 vises da s333769 er leder for laget og s333748 allerede er medlem.



Figur 6-19 En lagleder, et medlem og en invitert bruker på lagprofilen

#### 6.3.4.3 Kamphistorikk

En lagprofil har en kamphistorikk som gjør det mulig for en besøkende bruker, eller medlemmer av laget, å se de fem forrige resultatene laget, tilhørende lagprofil, har hatt. Lagprofilen i Figur 6-20 inkludere en kamphistorikk. Hver kamp i kamphistorikken inneholder turneringen kampen ble spilt i, og begge lagene i kampen med tilhørende rundepoeng. En vunnet kamp symboliseres med grønn farge, og en tapt kamp med rød. Ved å klikke på en kamp i kamphistorikken blir man videresendt til kampens kamprom.



Figur 6-20 Lagprofil med bilder, medlemmer, brukerinvitasjon og kamphistorikk

#### 6.3.4.4 Refleksjon over kravspesifikasjonen

I den originale kravspesifikasjonen var det spesifisert at lagprofilen skal ha en oversikt over alle brukerne som er medlemmer. Det ble også spesifisert at et lag skal ha en lagleder som har mulighet til å invitere spillere, fjerne spillere og bytte profilbilde/banner på profilsiden. Disse kravene er oppfylt.

Kravspesifikasjonen spesifiserte også at en lagprofil skulle ha alle tidligere resultater til et lag på en kamphistorikk. Under et sprintmøte med produkteier, ble det enighet om at vi bare skulle inkludere nylige resultater i kamphistorikken. En fullstendig utlisting av alle kamper på en egen side ble diskutert som noe ekstra som kunne bli lagt til ved Accenture sin videreutvikling av turneringsplattformen etter bachelorgruppen er ferdige med prosjektet. Det ble altså gjort en forandring etter produkteierens ønske, og dette kravet er oppfylt.

## 6.3.5 Oppretting av turnering

Alle innloggede brukere kan opprette en ny turnering. Da trykker man på «Create tournament»-knappen i navbaren. Da må man gjennom en registreringsprosess på fire steg.

Vi ønsket å dele opp registreringsprosessen for å ikke skape for mye kognitive belastning for brukeren ved å presentere store mengder informasjon på en gang. For å holde på brukeren sin oppmerksomhet, og for å holde han/hun oppdatert på hvor i prosessen man er, så ble det implementert en “progress bar”. Denne vises først i andre steg i registreringsprosessen som vist i Figur 6-22.

### 6.3.5.1 Første steg i registreringsprosessen

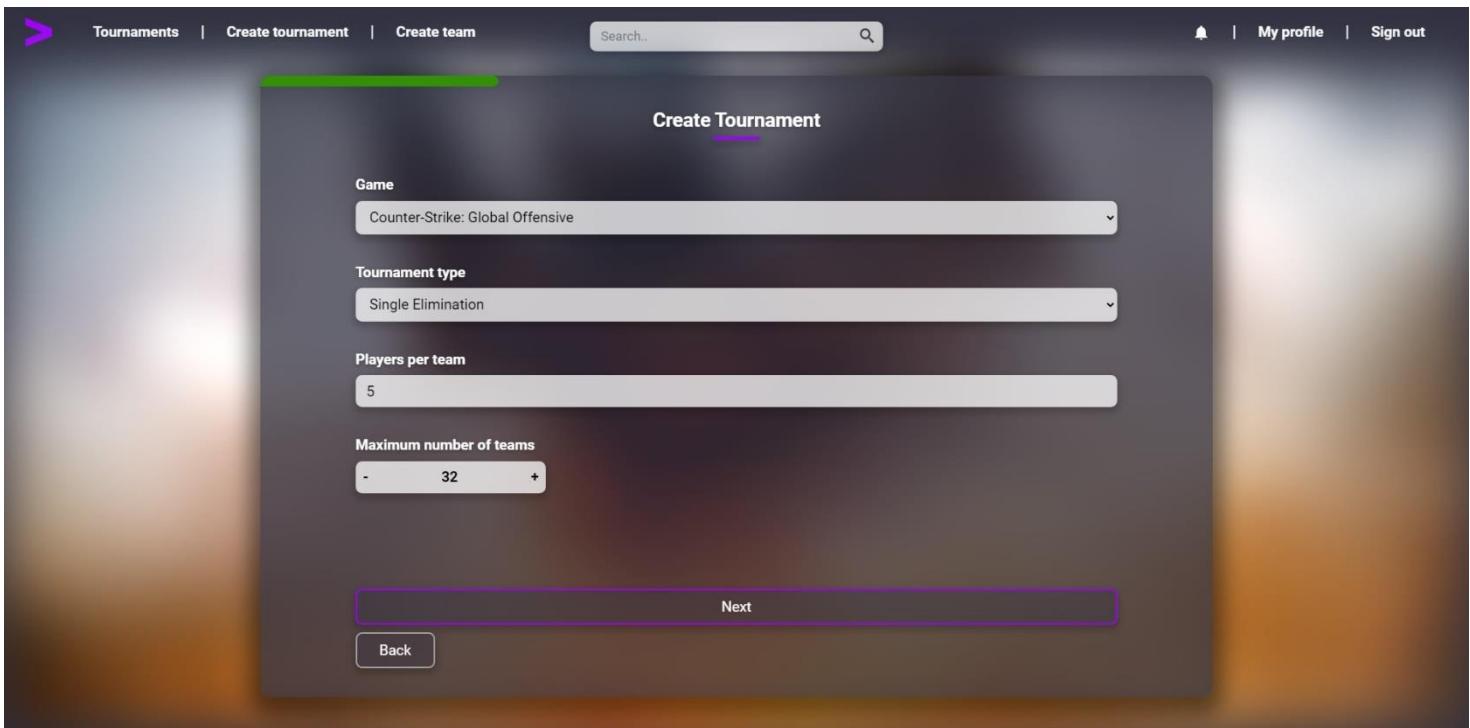
I første steg av registreringsprosessen må en bruker fylle inn praktisk informasjon rundt turneringen. Dette inkluderer turneringens navn, starttidspunkt og eventuelt land/område om dette skal spesifiseres. Om man ikke velger noe spesifikt land/område gjelder turneringen automatisk for alle land. Man har også muligheten til å legge til en beskrivelse på opptil 250 tegn. Dette vises i Figur 6-21.

The screenshot shows a web-based tournament creation interface. At the top, there are navigation links: 'Tournaments' (with a purple arrow icon), 'Create tournament' (which is circled in red), 'Create team', and a search bar with placeholder text 'Search..'. To the right are 'My profile' and 'Sign out' links. Below the header is a dark-themed form titled 'Create Tournament'. The form fields include: 'Title' (input: 'OsloMet Student Showdown'), 'Short description (optional)' (input: 'En hyggelig turnering for studenter på OsloMet. En god mulighet til å knytte nye bånd gitt tiden vi er i.'), 'Start time' (input: '05/26/2021 12:00 PM'), and 'Country / Area' (input: 'Norway'). A large purple-bordered button at the bottom is labeled 'Next'.

Figur 6-21 Første steg i prosessen av å opprette en turnering, og plassering av "Create tournament" knappen

### 6.3.5.2 Andre steg i registreringsprosessen

I andre del av registreringsprosessen må en bruker fylle ut med mer teknisk informasjon relatert til turneringen. Dette inkluderer hvilket spill som skal spilles, hva slags format turneringen skal ha, hvor mange spillere som skal være på hvert lag og hvor mange lag som skal få lov å være med. For øyeblikket er bare «Single elimination» implementert. Når man skal velge spill er også «Other games» en mulighet. Dette gjelder om man ønsker å holde en turnering i et spill som ikke er tilgjengelig. Andre steg i registreringsprosessen vises i Figur 6-22.



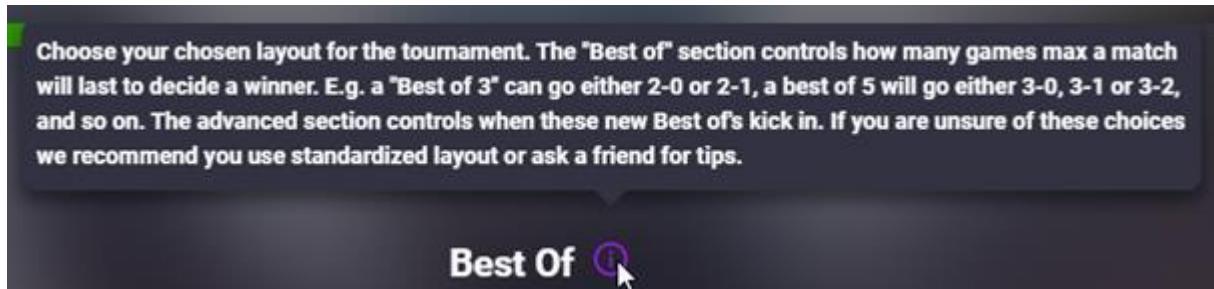
Figur 6-22 Andre steg i turneringsprosessen

### 6.3.5.3 Tredje steg i registreringsprosessen

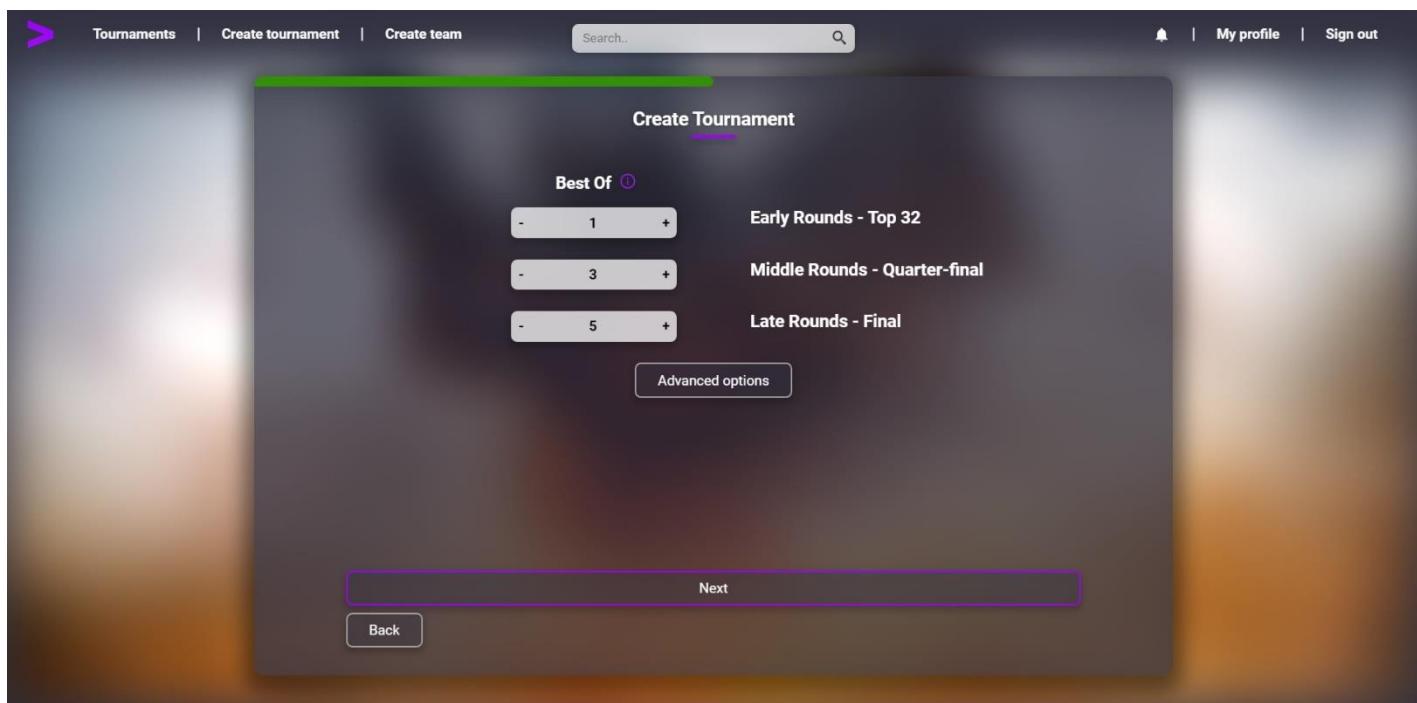
I det tredje steget av registreringsprosessen må en bruker bestemme hva slags best-av-format de forskjellige intervallene med runder i turneringen skal ha. Dette kan for eksempel være at alle kampene fra og med første runde til semifinalen spilles som best-av-en, og fra og med semifinalen til turneringen er ferdig spilles alle kamper som best-av-tre. En best-av-tre her vil ikke si at det er førstemann til å få to poeng i spillet som blir spilt, men førstemann til å vinne to fulle kamper i spillet.

Best-av-formatet blir justert gjennom et verktøy på siden, som vises i Figur 6-24. Det inneholder to kolonner, med tre rader. Den første raden lar en bruker justere hvilket best-av-format som skal bli brukt fra og med første kamp, frem til kvartsfinalen. De to neste kolonnene har samme funksjonalitet. Den andre kolonnen gjelder fra og med kvartsfinale frem til finalen, og den siste kolonnen for finalen. Det høyeste best-av-formatet som er implementert er best-av-sju, etter produkteiers ønske.

Siden best-av-verktøyet kan være vanskelig å forstå for uerfarne brukere valgte vi å implementere en «tooltip<sup>1</sup>» med forklarende tekst. Dette er vist i Figur 6-23.



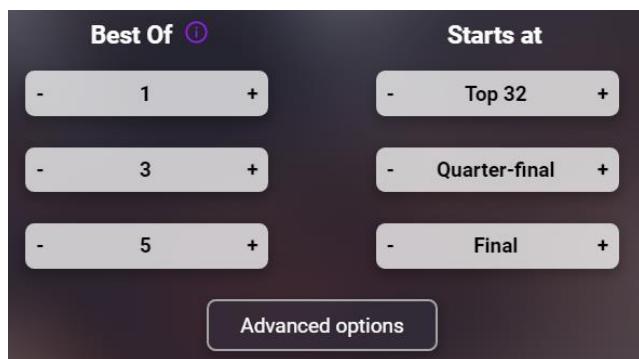
Figur 6-23 "Tooltip" med forklarende tekst, ment for nye brukere



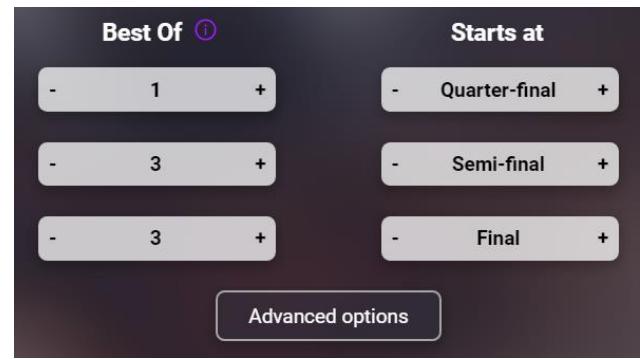
Figur 6-24 Verktøy som lar en bruker justere best av formatet gjennom turneringen

<sup>1</sup> Se Ordlisten 9

Best-av-verktøyet har også en «Advanced options» knapp ment for brukere som ønsker mer frihet til å justere best-av-formatene i turneringen. Når man klikker på denne gjøres høyre kolonne til knapper i lik stil som «Best of»-knappene vist på bildet over. Da kan man selv justere intervallet med kamper, som følger et best av format. Eksempler på dette vises i Figur 6-25 og Figur 6-26.



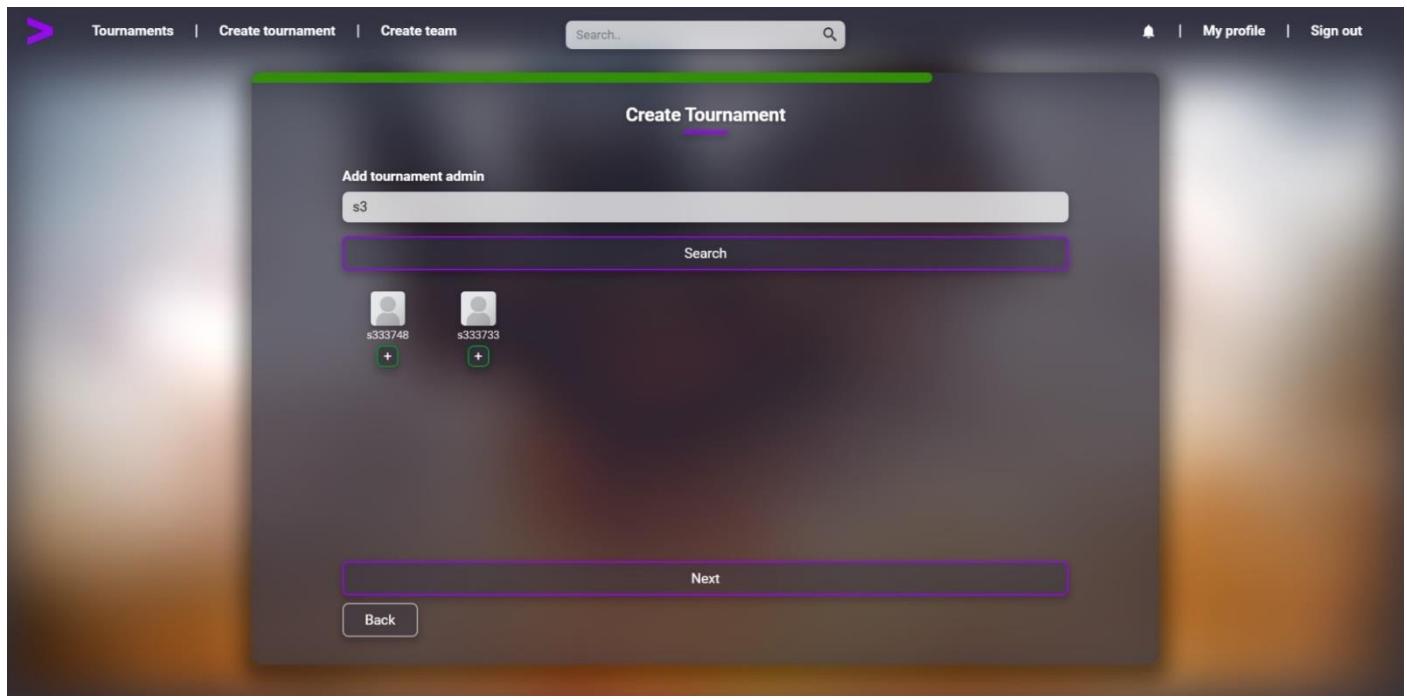
Figur 6-26 Best av en fra første kamp til kvartfinale, best av tre fra kvartfinale til finalen, og best av frem i finalen.



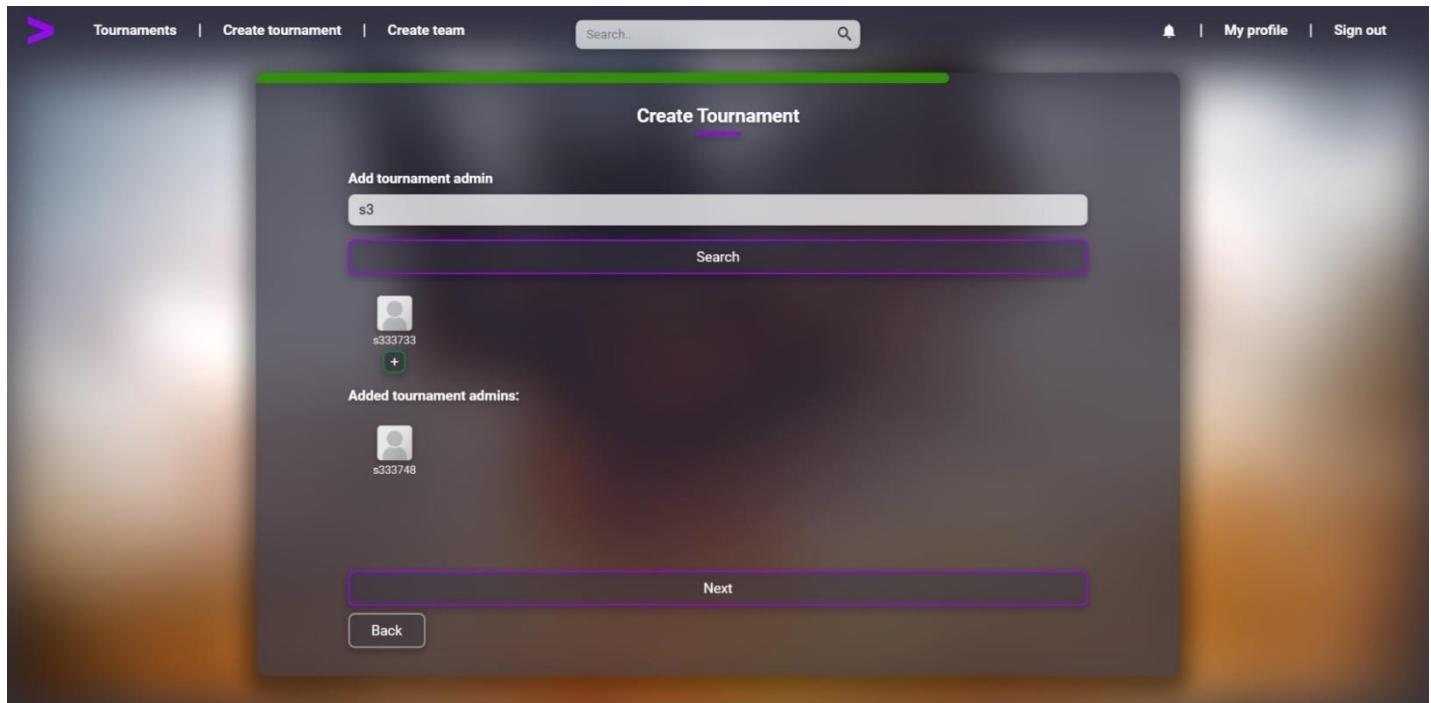
Figur 6-25 Best av en frem til kvartfinalen, best av tre fra semifinale til finalen er over.

#### 6.3.5.4 Fjerde steg i registreringsprosessen

I det siste steget i registreringsprosessen har en bruker mulighet til å legge til ekstra administratorer. Da bruker man en liknende søke-komponent som når man inviterer en spiller til et lag. Denne vises i Figur 6-27 og Figur 6-28**Error! Reference source not found..** Brukeren som oppretter turneringen, blir alltid lagt til som administrator.



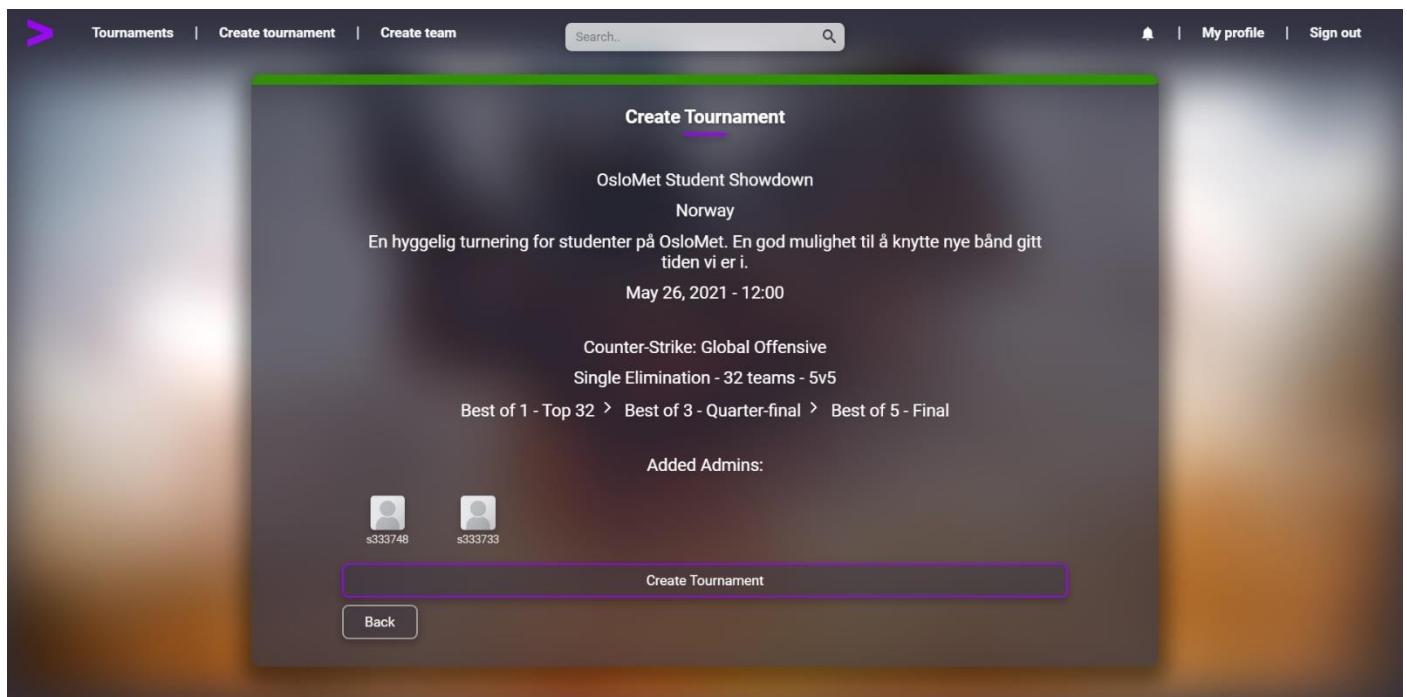
Figur 6-27 Bruker har ikke invitert noen administrator



Figur 6-28 Bruker har invitert en administrator

### 6.3.5.5 Kvittering for registreringsprosessen

Etter å ha konfigurert turneringen ferdig får man se en «turneringskvittering» som man kan bruke for å se over alt man har fylt inn i registreringsprosessen, før man velger å lage den. Dette vises i Figur 6-29.



Figur 6-29 Turneringskvittering

### 6.3.5.6 Refleksjon over kravspesifikasjonen

Kravspesifikasjonen sier at: «Innloggede brukere skal ha mulighet til å opprette turneringer. Ved oppretting av en ny turnering blir turneringsformat og en rekke andre parametere satt.» Dette er vagt og åpent for tolkning, og vi har derfor samarbeidet mye med produkteier gjennom utviklingen av registreringsprosessen for å forsikre oss om at det vi leverer samsvarer med hans ønsker.

### 6.3.6 Turneringsoversikt

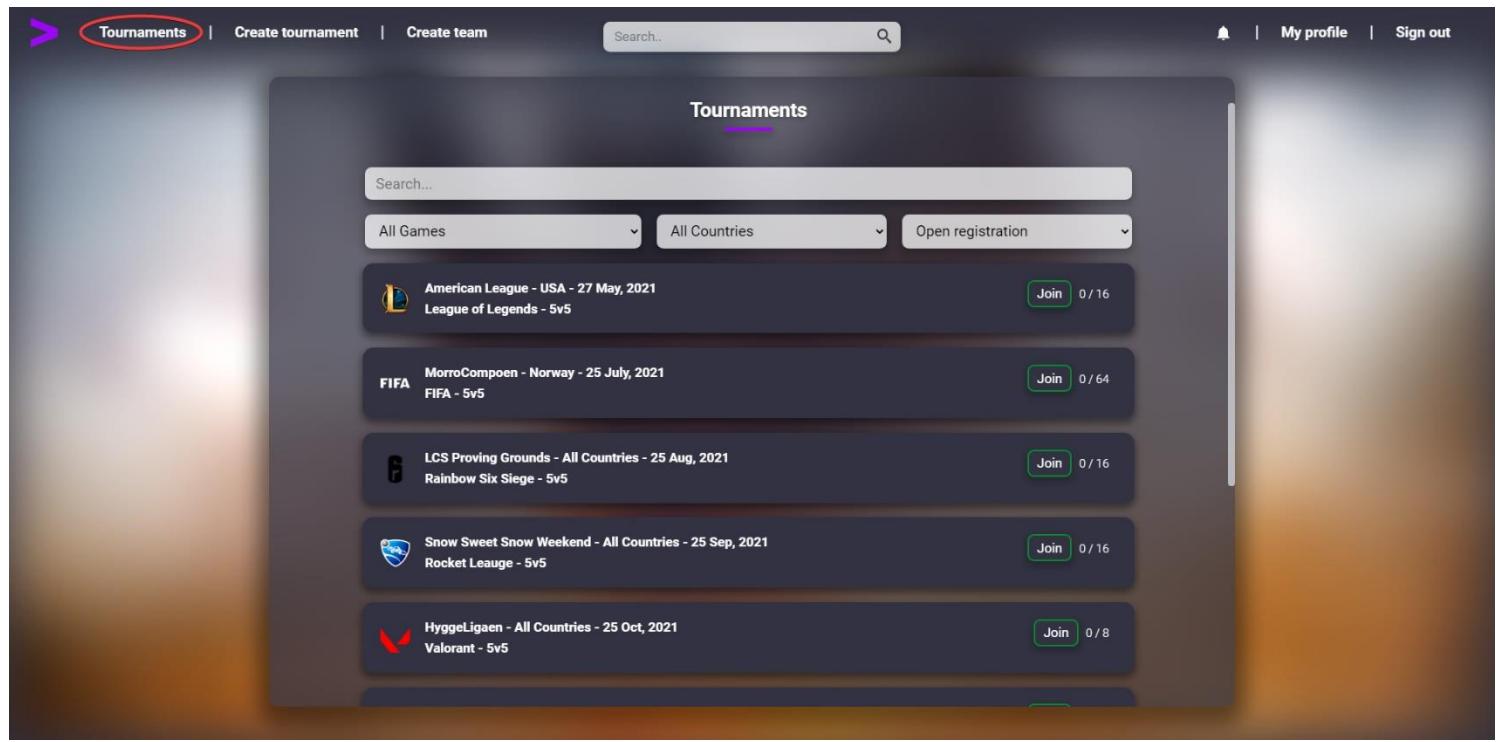
For å finne frem til turneringen man ønsker å spille skal det være en lett tilgjengelig turneringsoversikt på plattformen. Vi har derfor valgt å plassere lenken dit som første element i navbaren. Turneringsoversikten vises i Figur 6-30.

Øverst i turneringsoversikten er det et stort søkefelt. Her kan en bruker søke etter en turnering, og da vil listen med turneringer dynamisk oppdatere seg for hvert nye tegn i søkefeltet. I tillegg er det mulig å filtrere basert på spill, land/område og status.

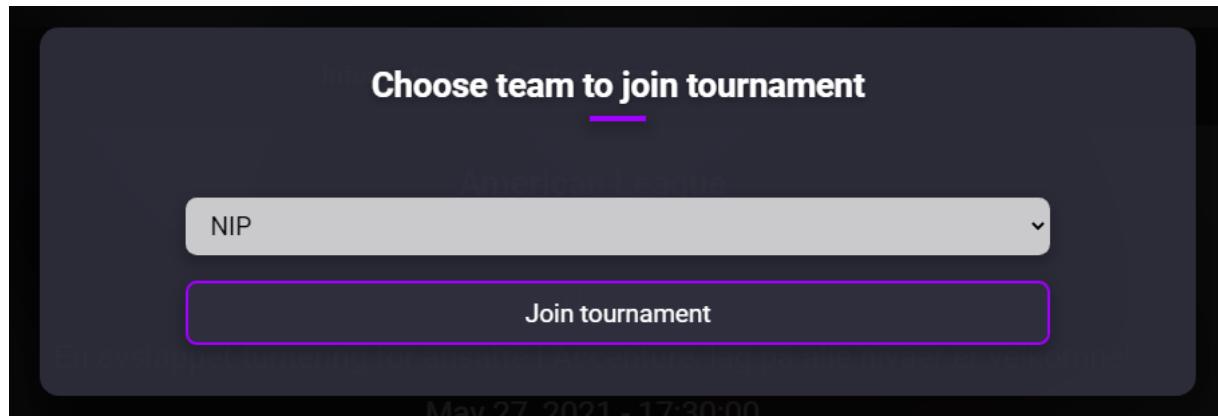
Det finnes fire statuser en turnering kan ha gjennom sin livssyklus. En turnering kan enten ha åpen påmelding, lukket påmelding, ha startet eller være ferdig. Mer om hvordan dette påvirker turneringsoversikten i neste avsnitt.

Turneringsoversikten er bygd opp av en liste med turneringskort som inneholder turneringens navn, land/område, dato, litt om turneringsinnstillingene og hvor mange som er påmeldt. Om påmelding fortsatt er åpen er det en grønn «Join»-knapp man kan trykke på for å melde på laget sitt. Dette vises i Figur 6-30. Da åpnes det et modalvindu med en nedtrekksmeny hvor man kan velge hvilke av lagene, man er leder for, man ønsker å melde på. Dette vises i Figur 6-31. Om turneringen har startet er det en rød visningsboks det står «Live» på for å vise at denne turneringen er i gang. Dette vises i Figur 6-32.

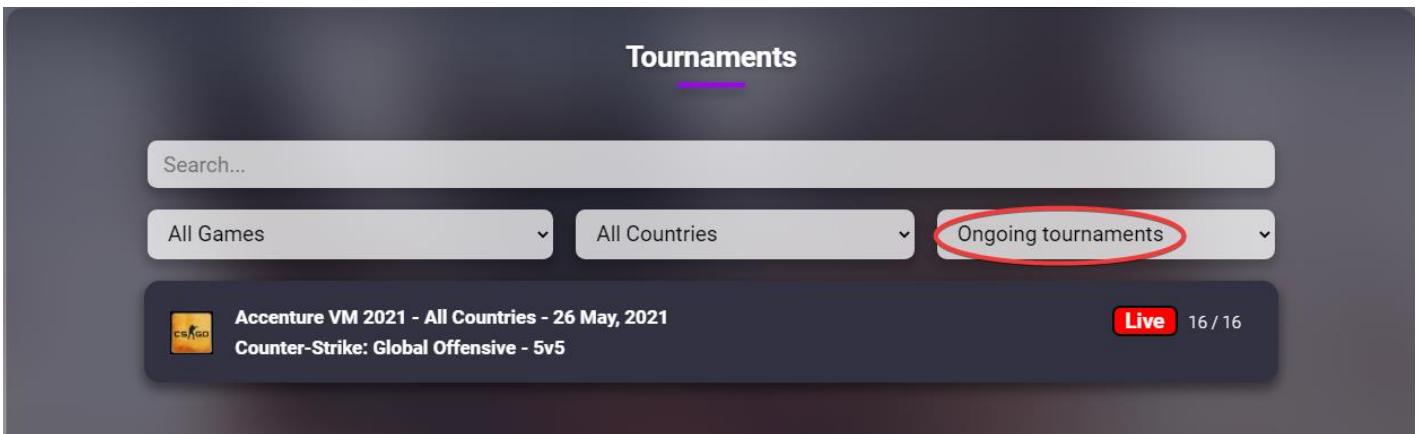
Kravspesifikasjonen spesifiserer at det skal være mulig å se tidligere, pågående og kommende turneringer. Kommende turneringer er turneringer som har åpen eller lukket påmelding. I tillegg ble det kravet «En turneringsoversikt skal kunne filtrere på land» lagt til underveis. Disse kravene innfridd. Skjermdump av turnerings-oversikten under.



Figur 6-30 Turneringsoversikten som viser kamper som fortsatt har åpen registrering



Figur 6-31 Modal-vindu for å bli med på turnering



Figur 6-32 Turneringsoversikt filtrert for pågående kamper

## 6.3.7 Turneringsside

### 6.3.7.1 Turneringens status

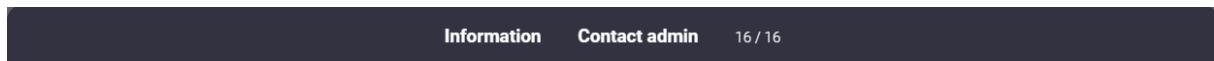
En turnering har fire statuser. Den kan ha en status av «0», som betyr at det er åpen påmelding. Om turneringens status er «1» betyr det at påmelding er lukket, men at turneringen fortsatt ikke har startet. Om turneringens status er «2» betyr det at turneringen har startet. Den siste statusen en turnering kan ha er «3». Dette betyr at turneringen er ferdig, og en vinner er kåret. Denne oversikten er også å finne i kapittel 8.2.1.

### 6.3.7.2 Navbar

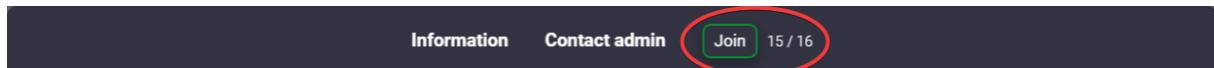
På turneringssiden blir en navbar brukt for å bevege seg mellom forskjellig innhold. Navbaren sitt innhold varierer basert på turneringens status.

### 6.3.7.2.1 Status «0»

Om turneringens status er «0» og turneringen ikke er full vil det ligge en grønn «Join»-knapp på navbaren. Når man trykker på denne åpner det samme modal-vinduet brukt for å bli med på turneringen som i turneringsoversikten. Dette vises i Figur 6-34. Om turneringens status er høyre enn «0», eller turneringen er full vil ikke denne lengre vises. Dette vises i Figur 6-33 og Figur 6-35.



Figur 6-33 Navbar tilhørende turnering med status høyre enn «0», eller maks antall lag påmeldt



Figur 6-34 Navbar tilhørende turnering med status «0», og med åpne plasser

### 6.3.7.2.2 Status «1»

Om turneringens status er «1» kommer navbaren til å se identisk ut som når status er «0», og det er maksimalt antall påmeldte lag. Se Figur 6-33.

### 6.3.7.2.3 Status «2»

Om en turnering har status «2» dukker det opp en ny overskrift på navbaren mellom «Information» og «Contact Admin». På denne overskriften står det «Bracket». Mer om bracketen senere. Dette vises i Figur 6-35.



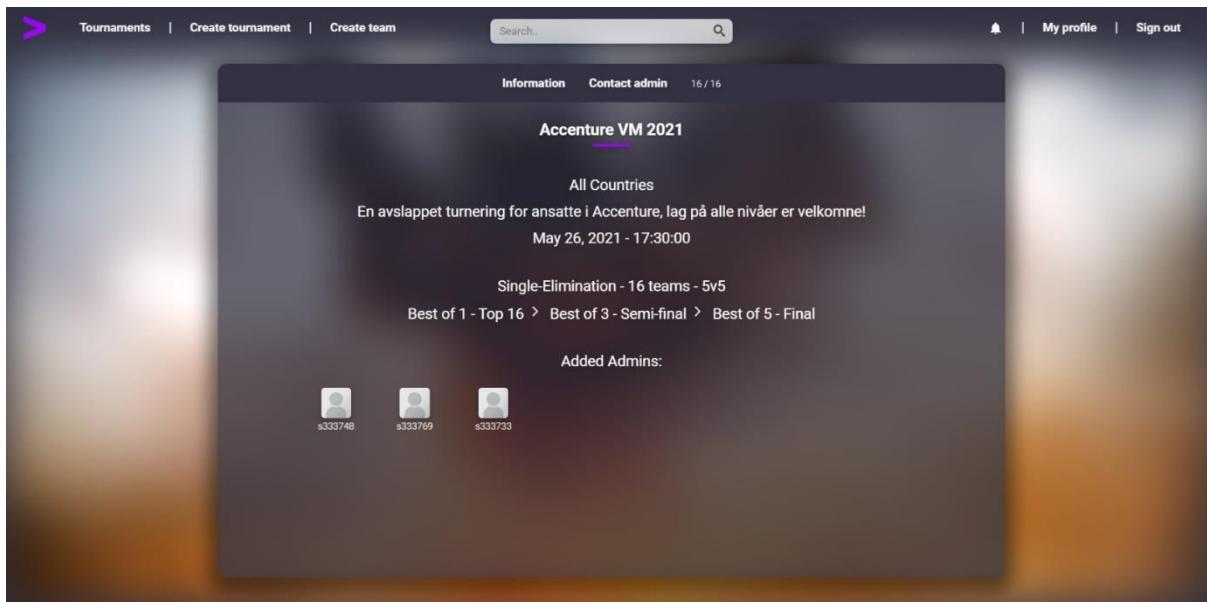
Figur 6-35 Navbar tilhørende turnering med status «2»

#### 6.3.7.2.4 Status «3»

Om en turnering har status «3» har den identisk navbar som når turneringen har status «2».

#### 6.3.7.3 Landingsside

Alle turneringer har en landingsside. Denne inkluderer informasjon i tilsvarende format som turneringskvitteringen man får når man oppretter en ny turnering. Denne siden ligger under «Information» på navbaren. Dette vises i Figur 6-36.



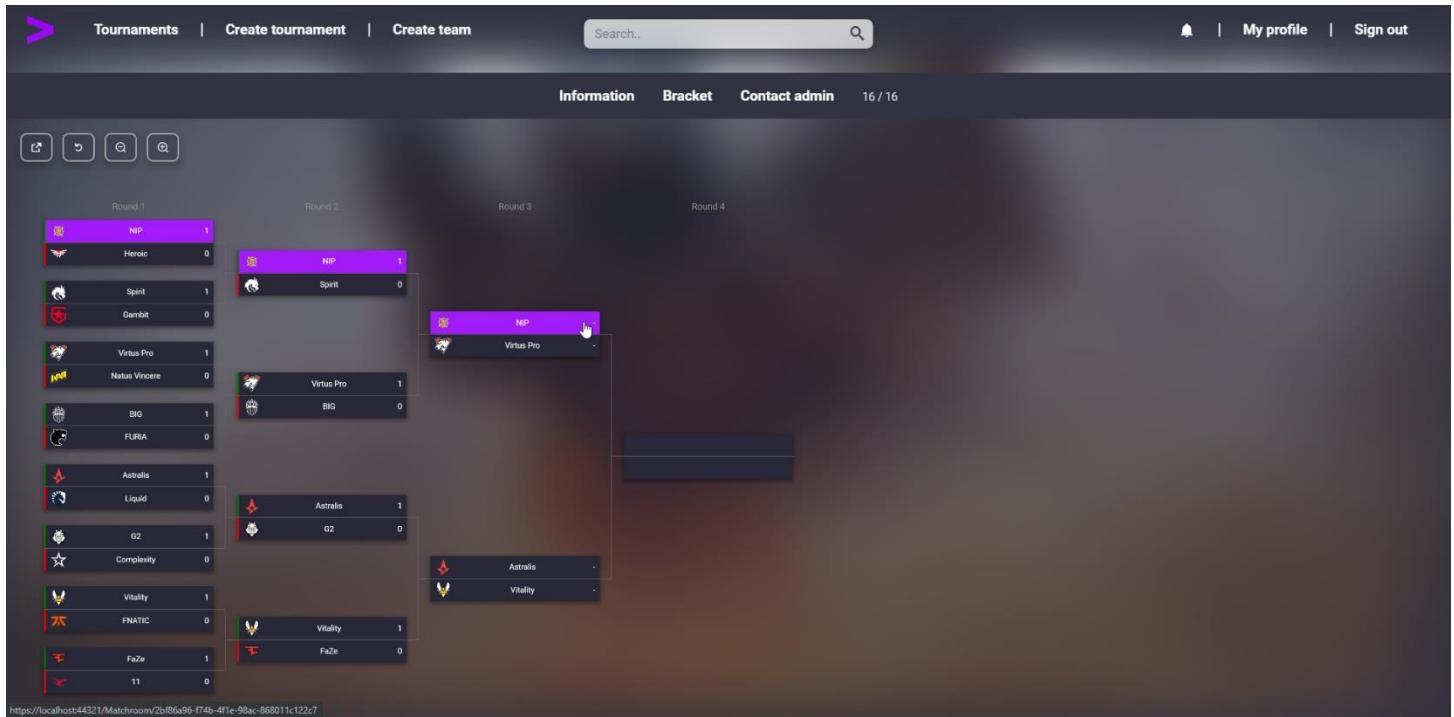
Figur 6-36 Turneringens landingsside

#### 6.3.7.4 Bracket

##### 6.3.7.4.1 Turneringstreet

Når en turnering starter, blir turneringstreet synlig for alle brukere. Turneringstreet ligger under «Bracket» i navbaren. For mer informasjon om hvordan street fungerer, se kapittel 6.4.4. Hver av streets noder er en kamp, og når man tar pekeren over et av lagene i en kamp får man se veien laget har tatt gjennom turneringen. Hver av

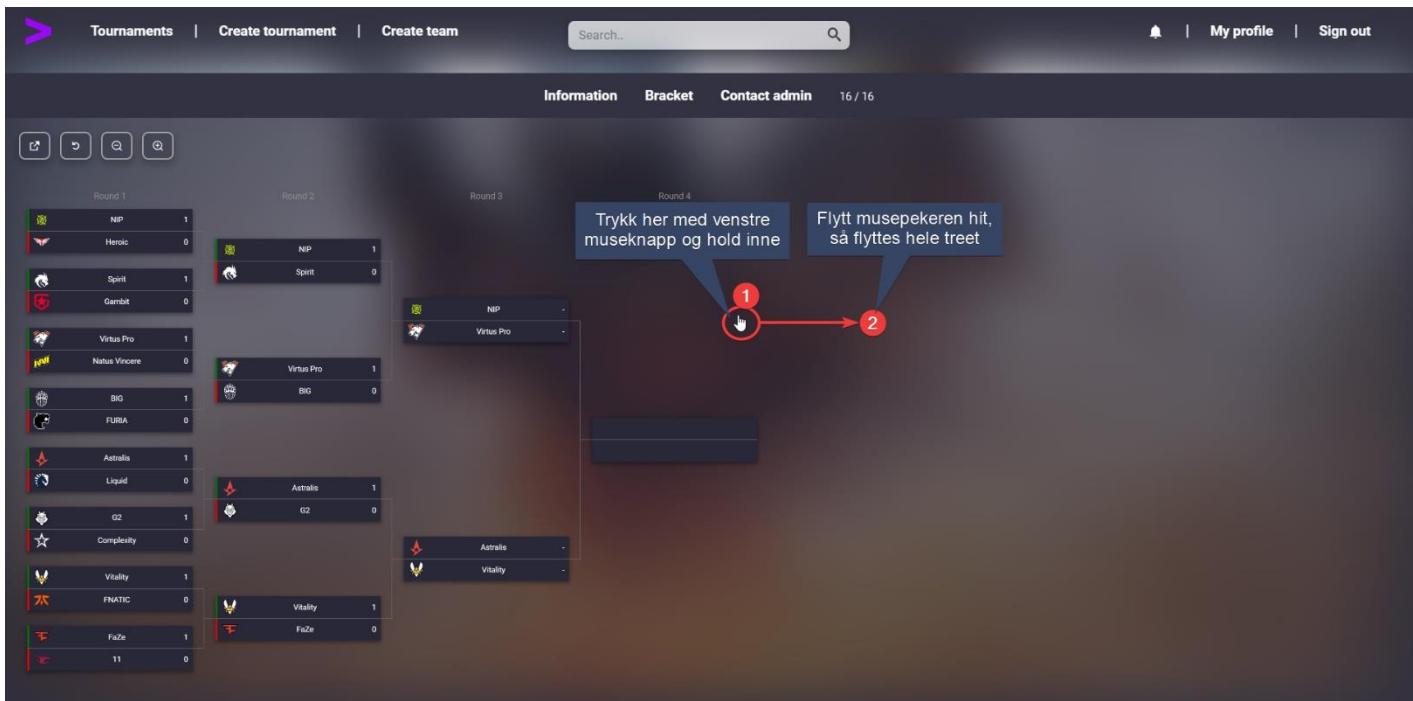
kampene består av laget, og laget sin logo. Om kampen er ferdig er vinner og taper markert med grønt og rødt, og stillingen til kampen. Om man trykker på et av lagene kommer man til tilhørende kampsida. Turneringstreet vises i Figur 6-37.



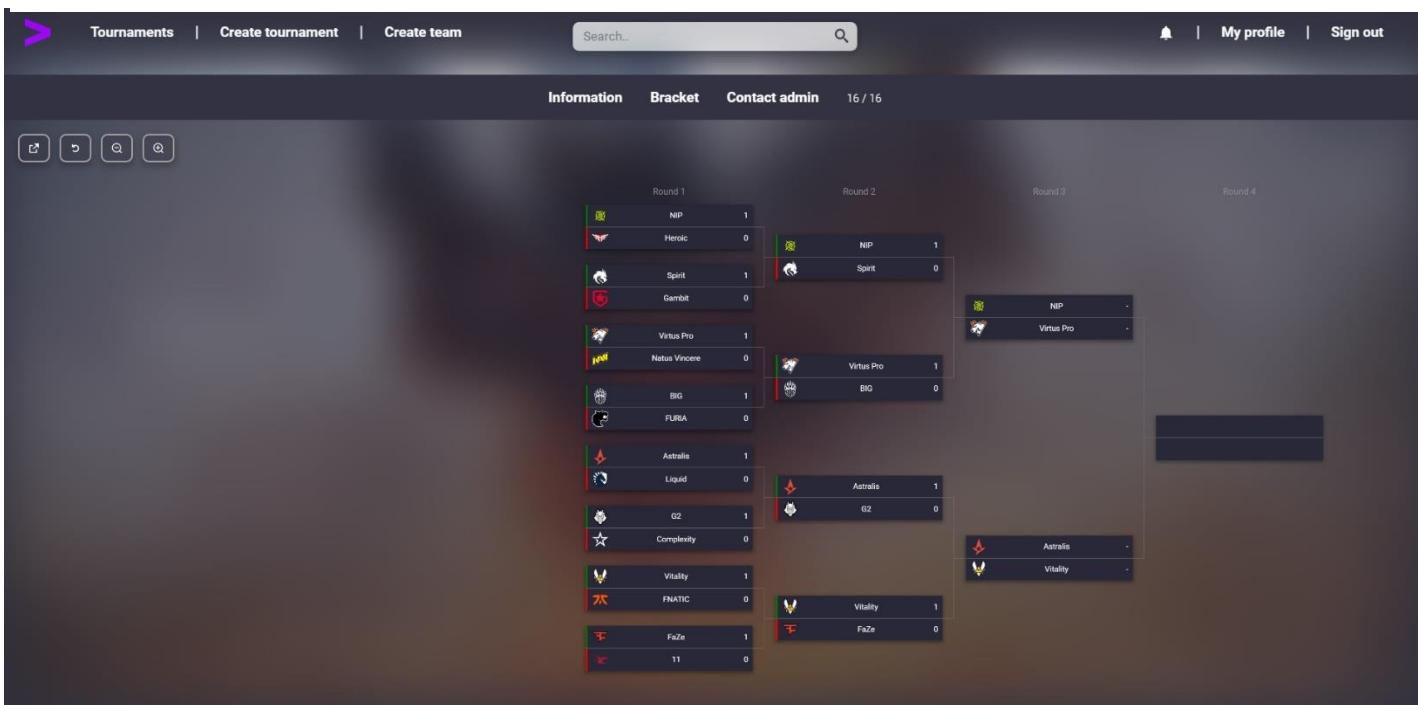
Figur 6-37 Turneringstreet

#### 6.3.7.4.2 Navigasjon i turneringstreet

En turnering kan inneholde store mengder lag. Da kan man bli nødt til å måtte traversere treet for å finne en kamp. For å gjøre dette kan man holde pekeren på bakgrunnen rundt turneringstreet, og holde inne venstre museknapp. Da tar man tak i innholdet, og man kan bevege treet vertikalt, eller horisontalt ved hjelp av å flytte datamusen. Dette vises i Figur 6-39.



Figur 6-39 Hvordan man beveger på turneringstreet



Figur 6-38 Turneringstreet etter å ha blitt flyttet

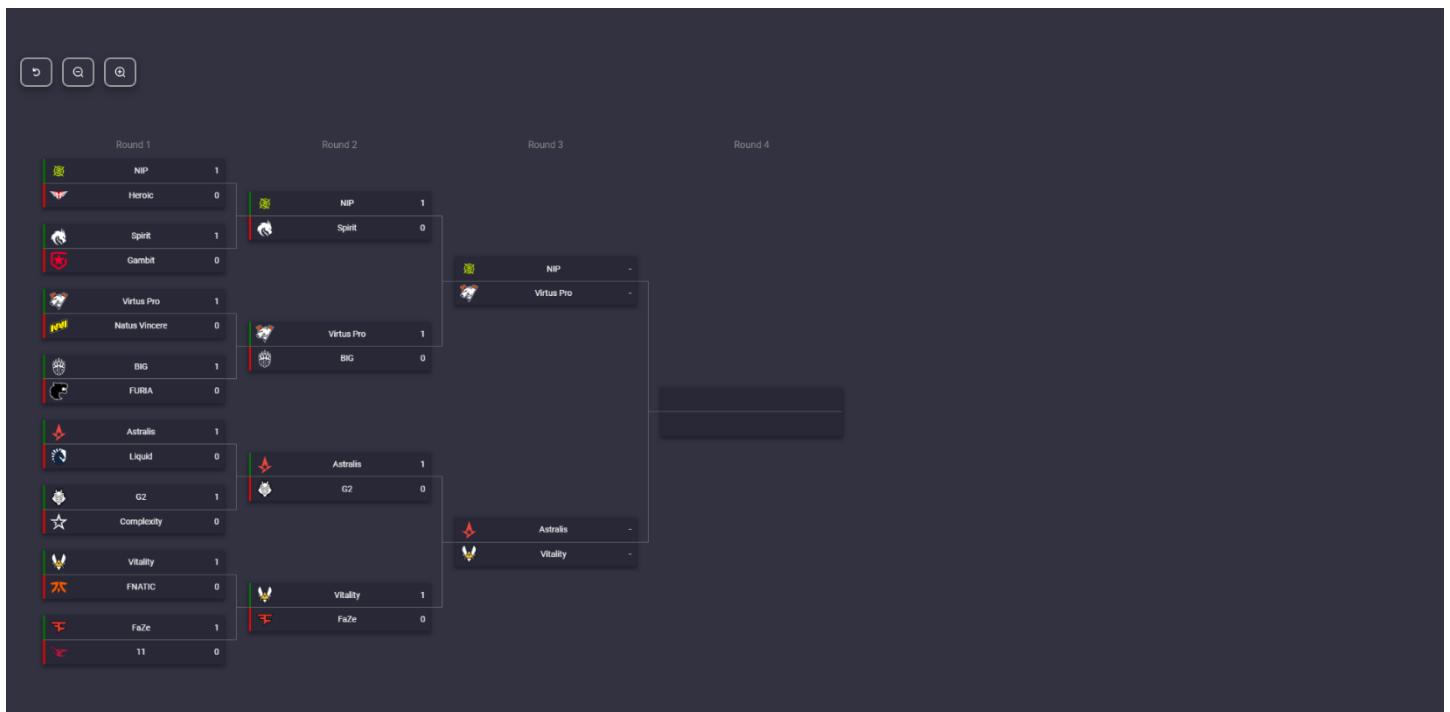
#### 6.3.7.4.3 Turneringsstreet funksjonsknapper

For å gjøre turneringstreet lettere å håndtere har vi implementert funksjonsknapper i tillegg til at treet er bevegelig. Det er fire funksjonsknapper. Disse vises i Figur 6-40.



Figur 6-40 Turneringstreets funksjonsknapper

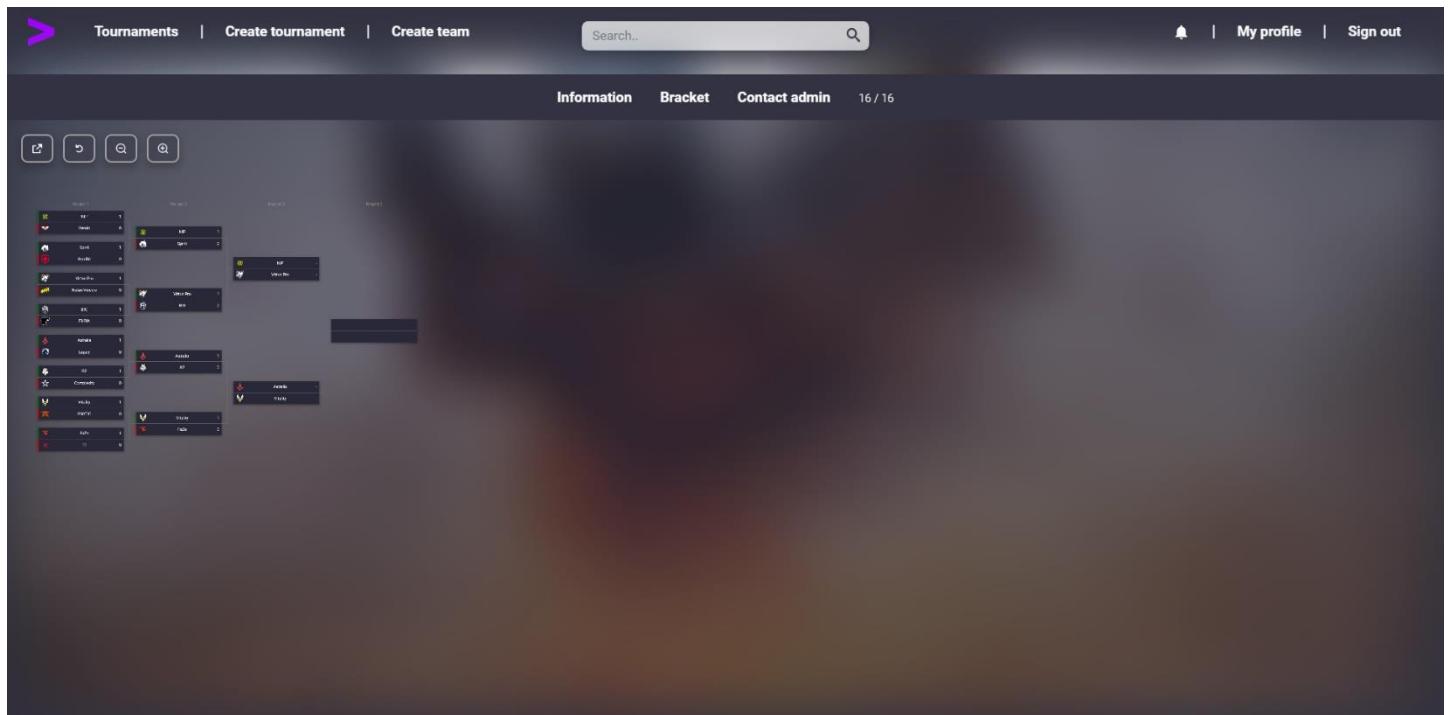
Den første funksjonsknappen brukes til å åpne en ny fane med turneringstreet i. Dette gjør større trær lettere å håndtere, da man har mer plass å bevege treet rundt på. Man kan også som deltager ha turneringstreet oppe i en annen fane om man ønsker å følge med på en annen skjerm. Turneringstreet i egen fane vises Figur 6-41.



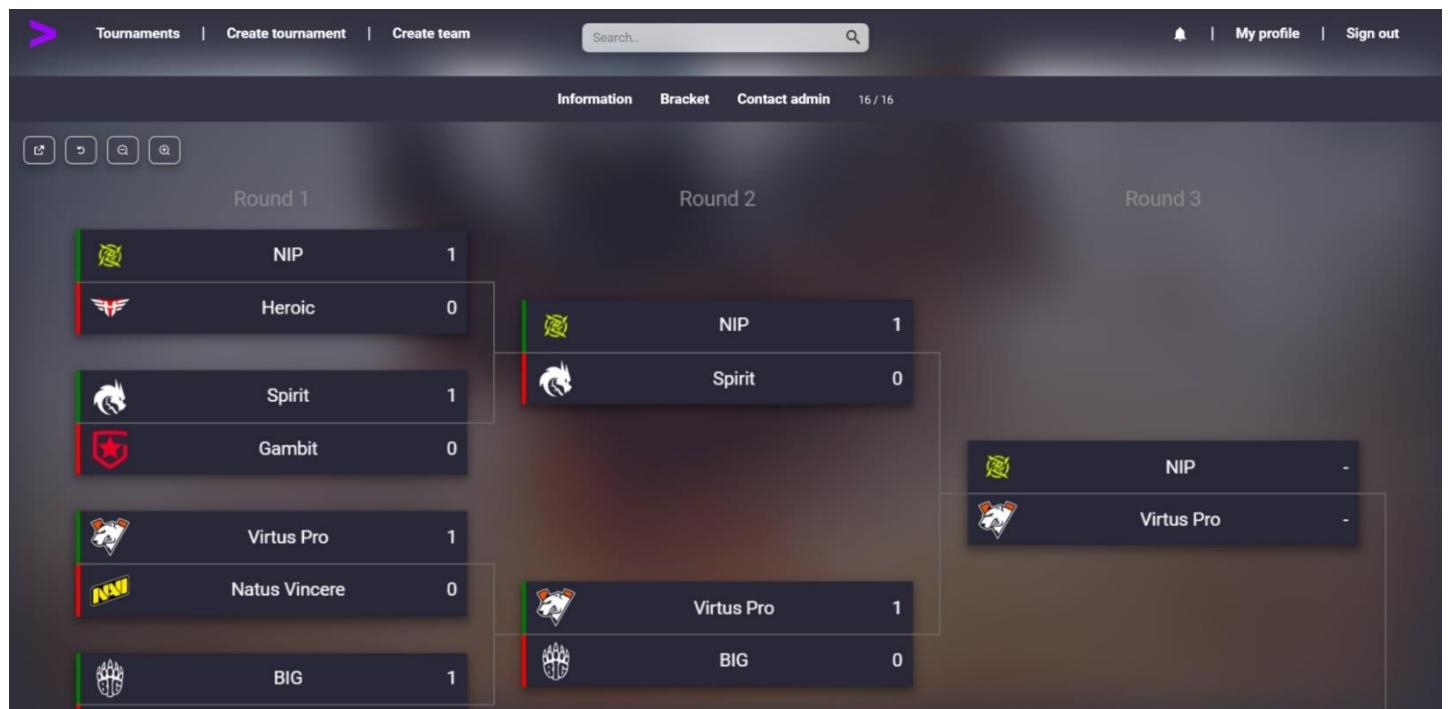
Figur 6-41 Turneringstreet i egen fane

Den andre funksjonsknappen er for å flytte treet tilbake til sin originale posisjon. Dette kan være nyttig om man har traversert et stort tre, hvor hele treet ikke har plass på brukerens skjerm. Dette er også nyttig om man har zoomet langt inn eller ut av treet. Mer om zooming i neste avsnitt.

Den tredje funksjonsknappen blir brukt til å zoome ut, og den fjerde til å zoome inn. Dette kan være nyttig om det er et veldig stort turneringstre. Da kan man zoome ut for å få bedre oversikt over treet, og deretter zoome inn der man ønsker. I tillegg til å bruke funksjonsknappene kan man også zoome inn og ut ved hjelp av å scrollle på musehjulet. Zooming vises i Figur 6-42 og Figur 6-43.



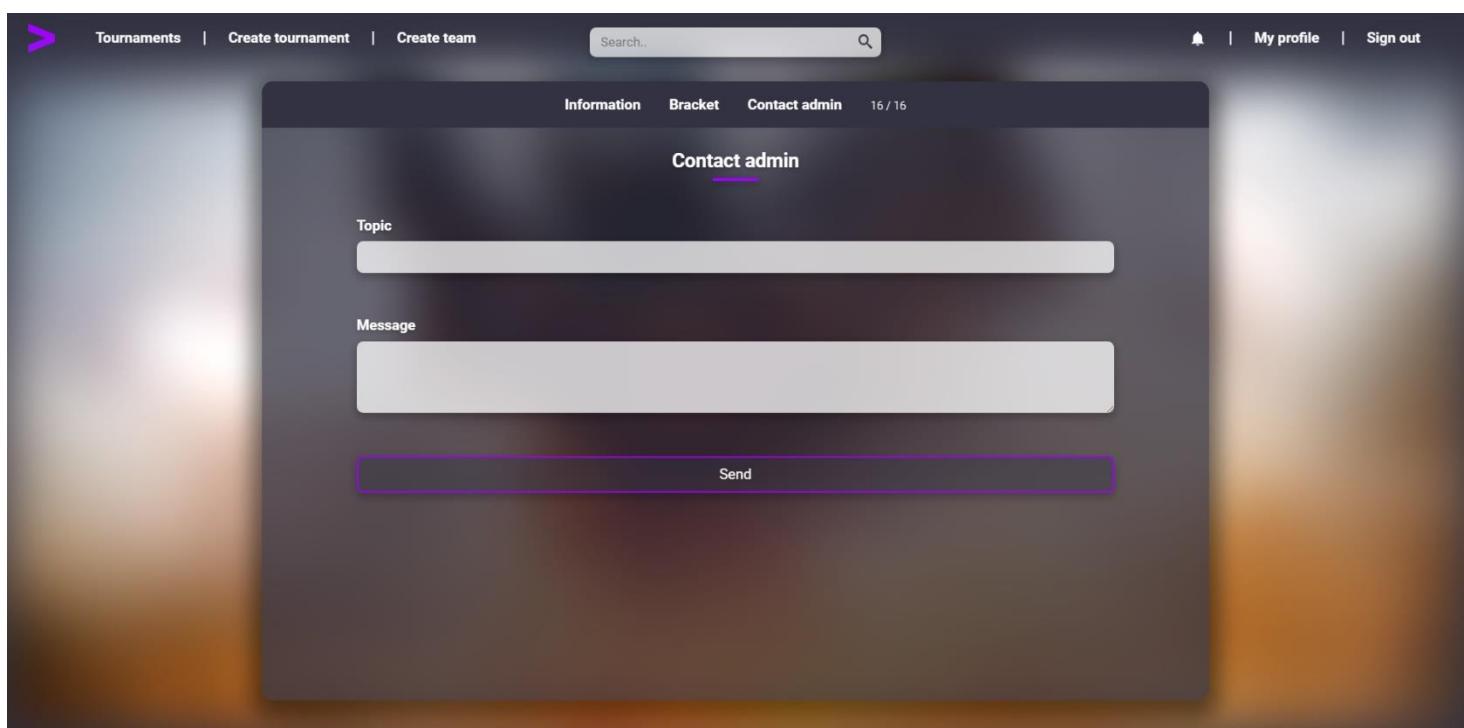
Figur 6-43 Turneringstre som har zoomet ut



Figur 6-42 Turneringstre som har zoomet inn

### 6.3.7.5 Kontakt turneringsadministrator

Som en innlogget bruker skal man ha muligheten til å kontakte turneringsadministrator. Dette kan gjøres ved å trykke på «Contact admin» overskriften i navbaren. Da forandrer innholdet på siden seg slik at man kan sende en felles mail til alle turneringsadministratorene, da alle brukere oppgir e-post ved registrering. På denne måten kan alle administratorene se om et spørsmål allerede har blitt besvart. Dette er bare mulig å gjøre for innloggede brukere. Siden for å kontakte turneringsadministrator(er) vises i Figur 6-44.



Figur 6-44 Vindu hvor en bruker kan kontakte turneringsadministratorene

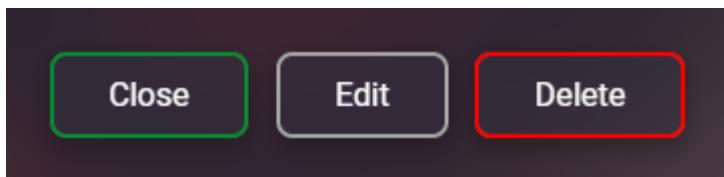
### 6.3.7.6 Refleksjon over kravspesifikasjonen

Den originale kravspesifikasjonen sier at «Plattformen skal ha en oversikt over tidligere, pågående og kommende turneringer, samt en side med informasjon for hver enkelt turnering.» Hvordan turneringssiden med informasjon skulle se ut, og hvordan

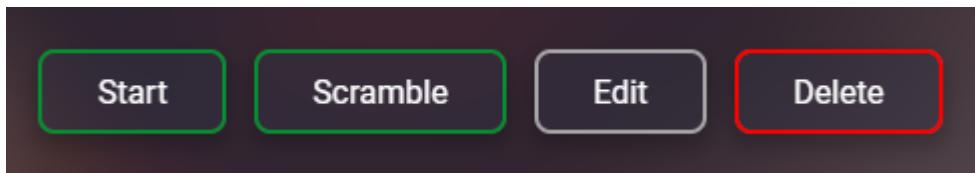
man gjennom denne skulle kunne navigere seg rundt i turneringen var åpent for tolkning, og vi har derfor, i samarbeid med produkteier, designet løsningen presentert over.

### 6.3.8 Turneringsside for turneringsadministrator

Som en turnerings-admin har man rettigheter utover det som gjelder for en vanlig bruker. Det inkluderer funksjonalitet for å stokke om på hvilke lag som møter hverandre i første runde, lukke påmelding, starte, redigere og slette turneringen. Redigering fungere på samme måte som registrering, med unntak av at alle inputfeltene er forhåndsfylt med den gjeldene turneringsinformasjonen. Det er noen av funksjonene som bare er tilgjengelig i deler av en turnering sin livssyklus. Det er ikke mulig å starte eller stokke før påmelding er lukket, og det er ikke mulig å stokke etter at turneringen er startet. De ulike funksjonsknappene vises på Figur 6-45, Figur 6-46 og Figur 6-47. Disse ligger på turneringens landingsside som vise i Figur 6-48.



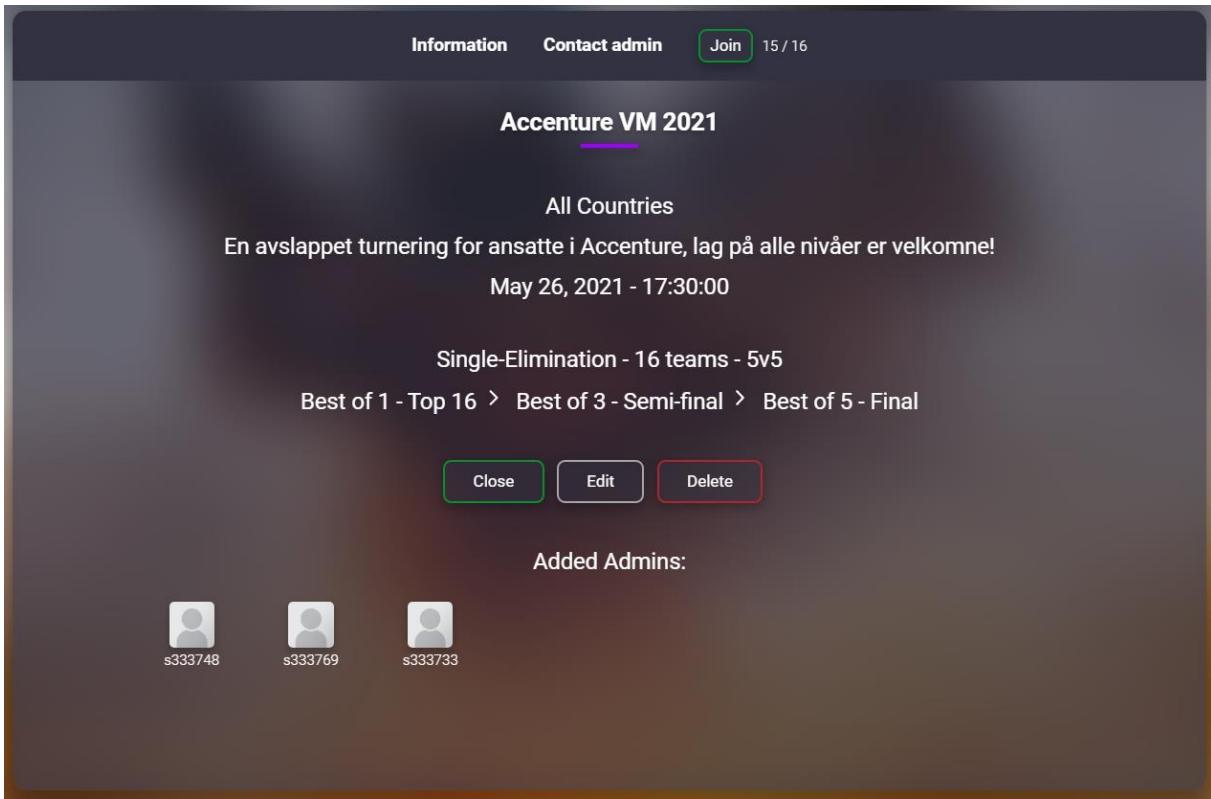
Figur 6-45 Funksjonsknappene for turneringsadministratoren, før påmelding er lukket



Figur 6-46 Funksjonsknappene for turneringsadministratoren etter påmelding er lukket



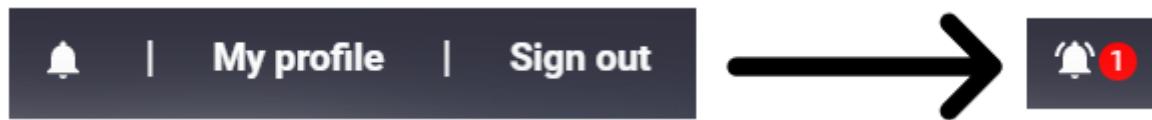
Figur 6-47 Funksjonsknapper for turneringsadministratoren etter at turnering er startet



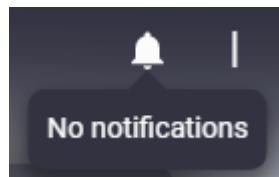
Figur 6-48 Funksjonsknapper på en turnerings landingsside for turneringsadministratorer

### 6.3.9 Varsel

Som innlogget bruker er det mulig å få to typer varsler. Informasjonsvarsel, og invitasjonsvarsel. Når man har fått et varsel forandres utseende på varselbjellen i navigasjonsbaren, og viser hvor mange varsler man har. Vi har valgt å gå med rød farge for dette, da rødt skiller seg ut fra designet vårt og tar oppmerksomheten til brukeren. Dette vises i Figur 6-49. Varsler blir levert i sanntid. Dette vil si at en bruker ikke behøver å oppdatere nettsiden for å få dem, men at varselet plutselig dukker opp på skjermen til brukeren. Dette er gjort gjennom teknologien SignalR, og blir beskrevet i nærmere detalj i kapittel 6.4.5.



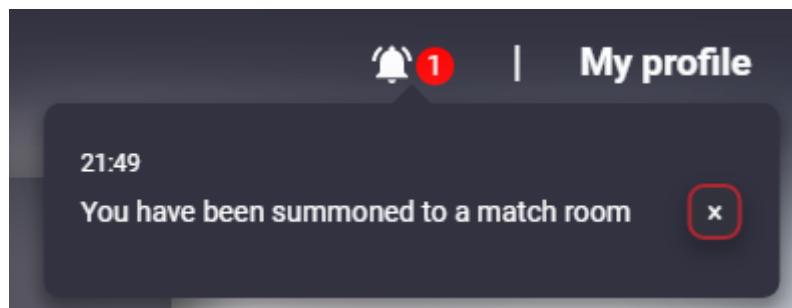
Figur 6-49 En bruker får et varsel



Figur 6-50 Tom varselliste

### 6.3.9.1 Informasjonsvarsel

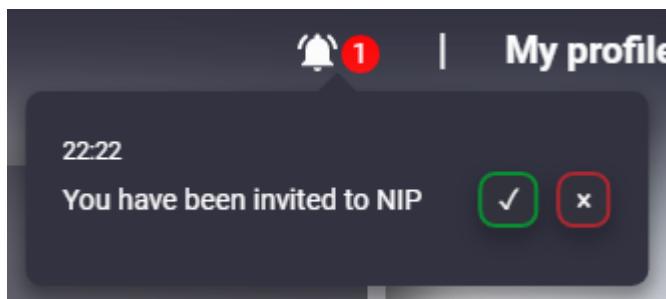
Informasjonsvarselet er det mest brukte varselet på turneringsplattformen. Det viser informasjon om at det har skjedd en oppdatering på en side som brukeren er knyttet til. Dette kan være ved at en kamp, hvor brukeren er en deltaker, har fått et resultat levert, eller ved at man er turneringsadministrator og blir tilkalt til et kampprom. Da kan man trykke på teksten i varselboksen, så blir man tatt til rommet varselet gjelder. I tillegg er det en rød knapp som blir brukt til å fjerne varselet. Informasjonsvarselet vises i Figur 6-51.



Figur 6-51 Eksempel på et informasjonsvarsel

### **6.3.9.2 Invitasjonsvarsel**

Et invitasjonsvarsel blir sendt ut når man inviterer en spiller til et lag. Da får denne spilleren et varsel som inneholder en kort tekst om hvilket lag som sendte invitasjonen, en knapp for å godta tilbudet, og en knapp for å avslå. Her kan man også trykke på teksten og da bli tatt til lagsiden tilhørende laget som sendte deg invitasjonen. Invitasjonsvarsel vises i Figur 6-52.



Figur 6-52 Invitasjonsvarsel til laget NIP

### **6.3.10 Kampprom og chat**

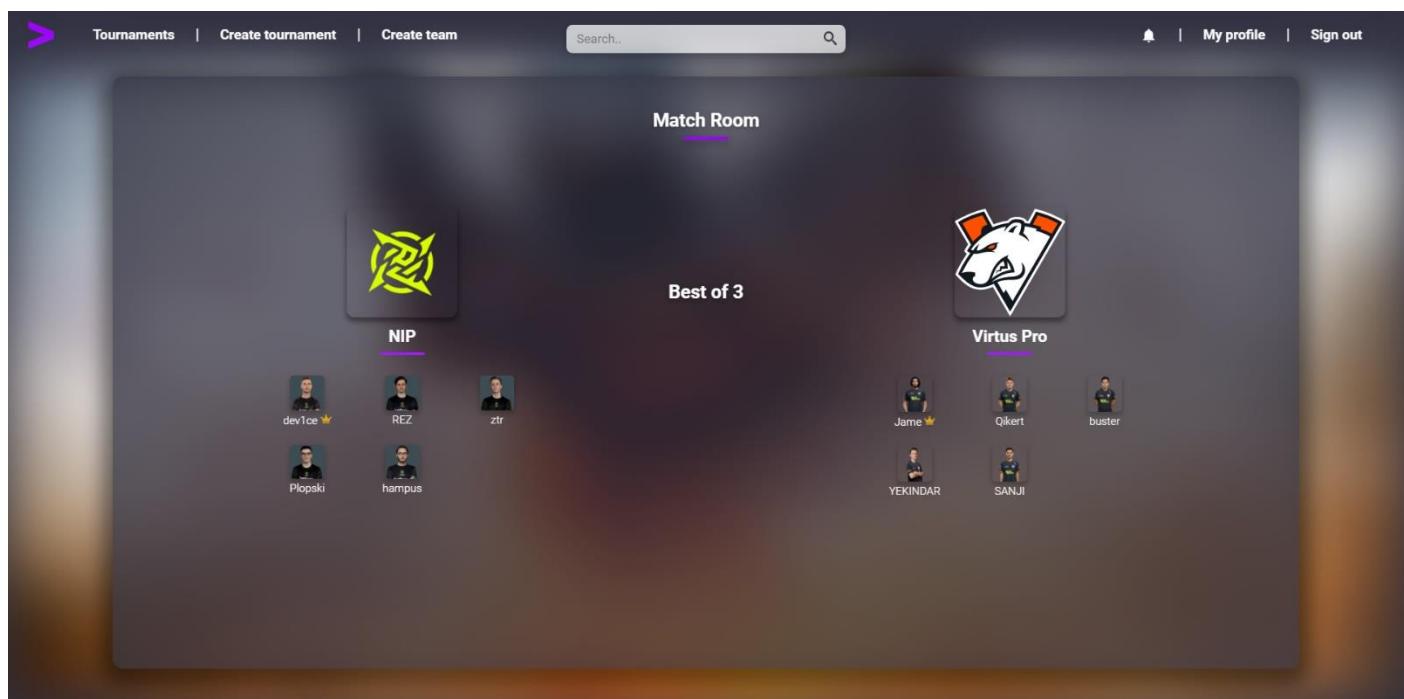
Alle kamper i en turnering har et eget kampprom. Dette er tilgjengelig for alle innloggede brukere, men innholdet kommer til å variere basert på en kamp sin status, og rollen til besøkende.

#### **6.3.10.1 Statuser**

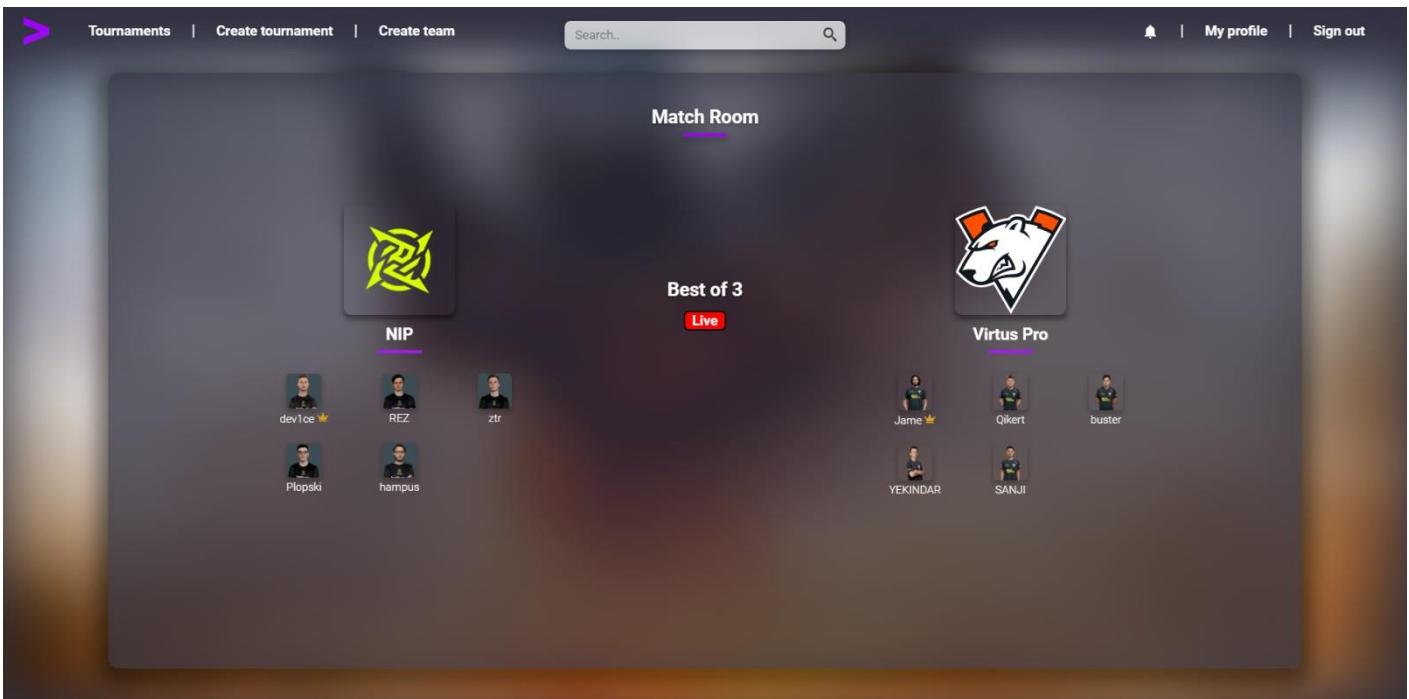
En kamp har tre forskjellige statuser som dikterer utseende til kampprommet. Når kampen ikke enda har startet er statusen satt til «0». Når en kamp har startet er statusen satt til «1». Når en kamp er ferdigspilt, er statusen satt til «2».

### 6.3.10.2 Kamprom for vanlig bruker

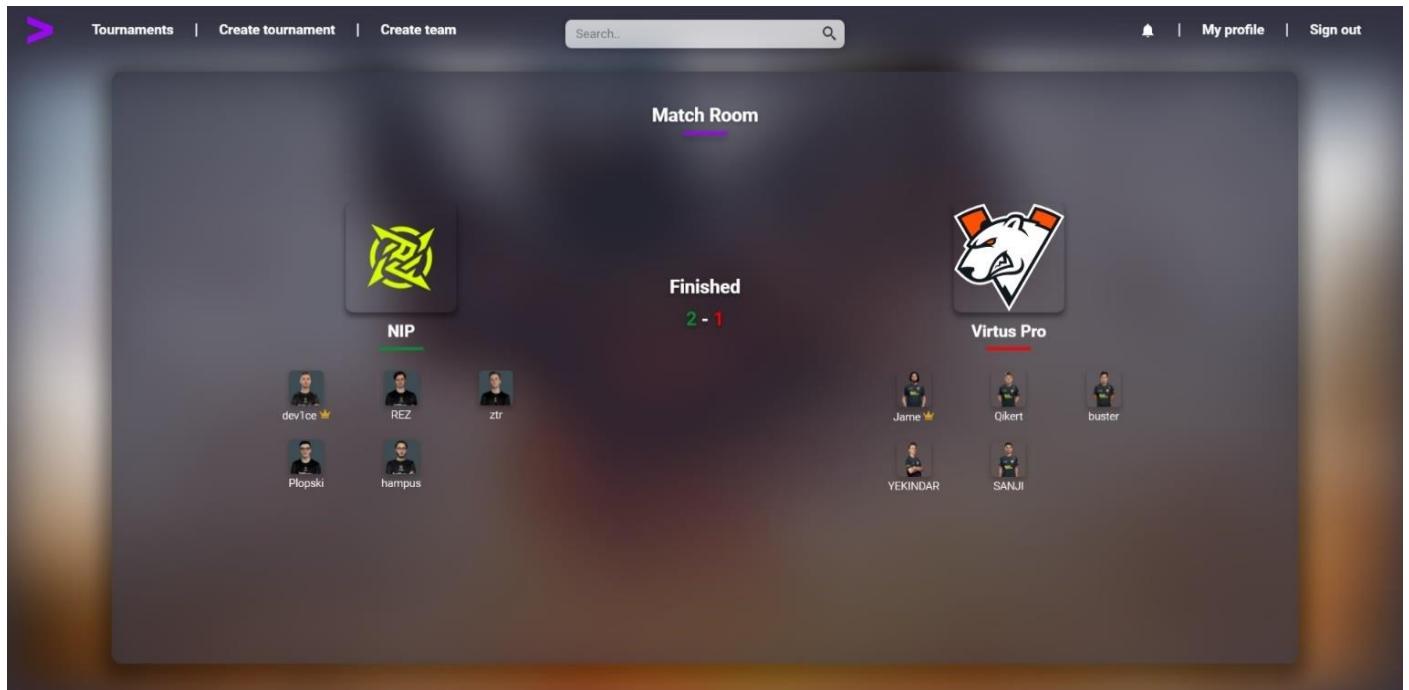
Alle brukere har tilgang til ethvert kamprom, men disse kamprommene har ikke noen nevneverdig funksjonalitet om man ikke er deltaker. De er hovedsakelig for å vise kampens lag, deltakere og status. Når prosjektet blir videreført innad i Accenture planlegges det å implementere statistikk fra spesifikke spill, og dette skal kunne bli sett av alle brukere. Vi har dermed lagt opp til videre utvikling av kamprommene for vanlige brukere ved å gi dem tilgang til en enkel visning nå som kan utvides i fremtiden. Dette vises i Figur 6-53, Figur 6-54 og Figur 6-55.



Figur 6-53: Kampron for vanlig bruker med status «0»



Figur 6-54 Kamprom for vanlig bruker med status "1". «Live» melding har blitt lagt til for å vise at kampen har startet



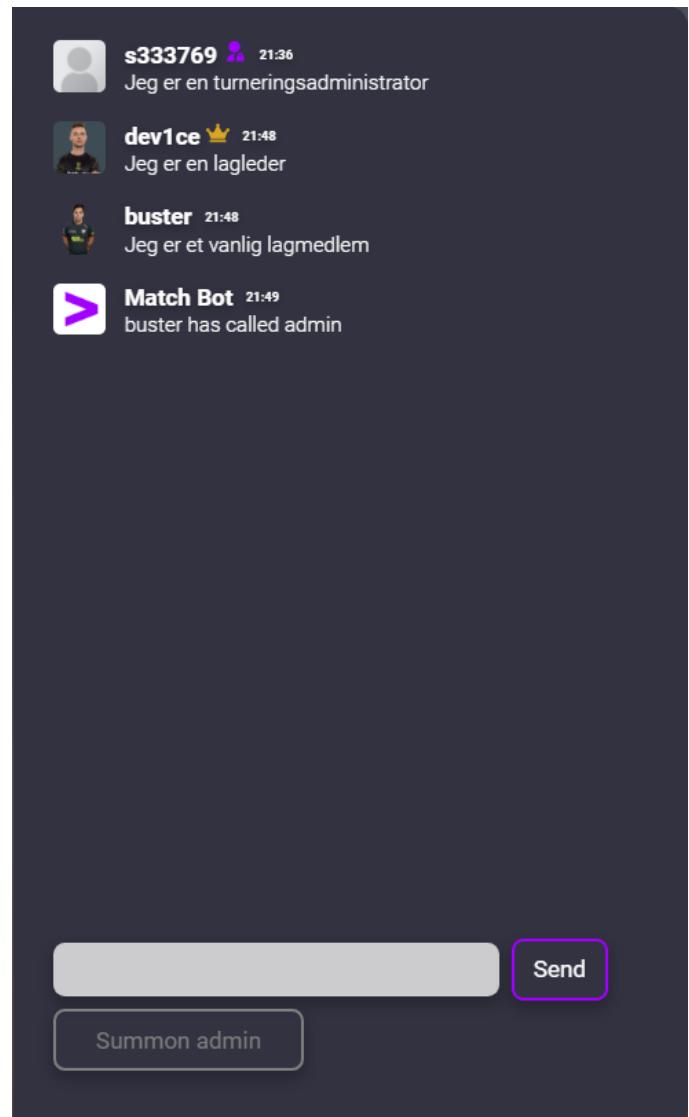
Figur 6-55 Kamprom for vanlig bruker med status "2". Resultatet vises, vinner symboliseres med grønt, taper med rødt.

### 6.3.10.3 Kameprom-chat

Alle kameprom har en chat som er synlig for kampdeltakere, og turneringsadministratorer. Gjennom denne kan deltakerne kommunisere med hverandre og med turneringsadministratorer. Lagledere og turneringsdeltakere har spesielle iconer i chatten som viser rollen deres. Dette vises i Figur 6-56.

I tillegg til å chatte med hverandre, kan man også tilkalle turneringsadministrator gjennom kampchatten. Dette gjøres ved å trykke på «Summon admin»-knappen. Da blir det sendt en melding i chat av «Match Bot» som informerer om hvilken spiller som har tilkalt administrator. Denne knappen blir deaktivert etter bruk, slik at man ikke kan spamme turneringsadministrator.

I likhet med varsler blir meldinger sent i sanntid. Dette er også blitt gjort gjennom SignalR, igjen kan man lese om dette i kapittel 6.4.5.

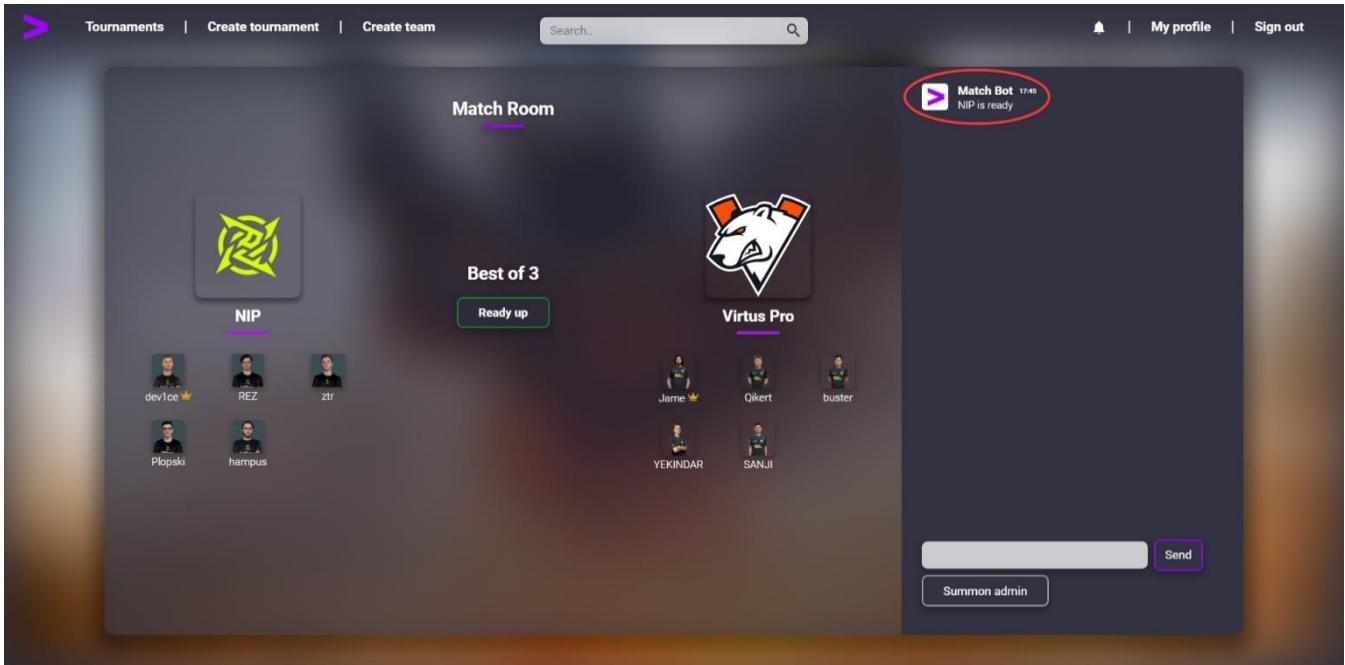


Figur 6-56 chatboks

### 6.3.10.4 Kameprom for kampdeltager

For en kampdeltaker har kameprommet ekstra funksjonalitet som en chat, og knapper som lar dem starte kampen og sende inn resultat. Når et lag blir flyttet inn i et kameprom er status automatisk «0», altså at kampen ikke har startet enda. Begge lagene har en knapp det står «Ready up» på. Når en lagkaptein trykker på denne så blir den forandret, og viser at laget har gjort seg klart. Dette vises i Figur 6-57. I tillegg sender «Match Bot» en melding i kampchatten, slik at motstanderne får beskjed om dette. Det

er bare lagledere som kan se, og klikke på «Ready up»-knappen. Kamprommet vises på figuren under



Figur 6-57 Kamprom med status "0" hvor motstanderlaget har gjort seg klare.

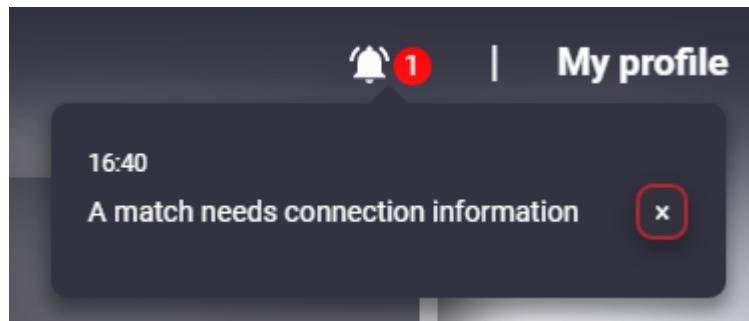


Figur 6-58 "Ready up" når den blir trykket på

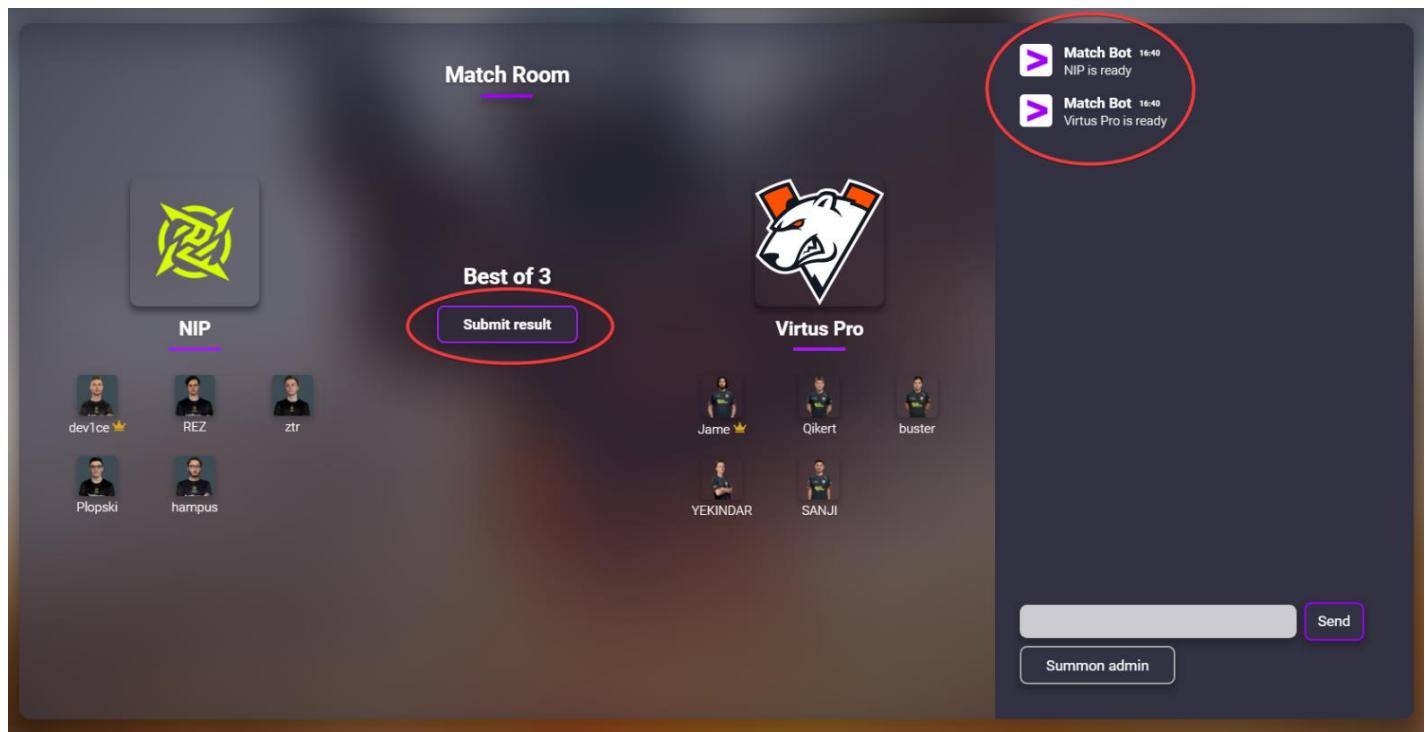
Når begge lagene har gjort seg klare blir kampens status satt til «1», og sidens innhold oppdaterer seg dynamisk for alle klientene som er inne på kamprommet.

Knappen det sto «Ready up» på blir gjort om til en «Submit result»-knapp. Dette vises i Figur 6-58. Mer om denne knappen i neste avsnitt. Turneringsadministratører får også

et informasjonsvarsel som sier at en kamp trenger tilkoblingsinformasjon for å kunne starte. Dette vises i Figur 6-60.



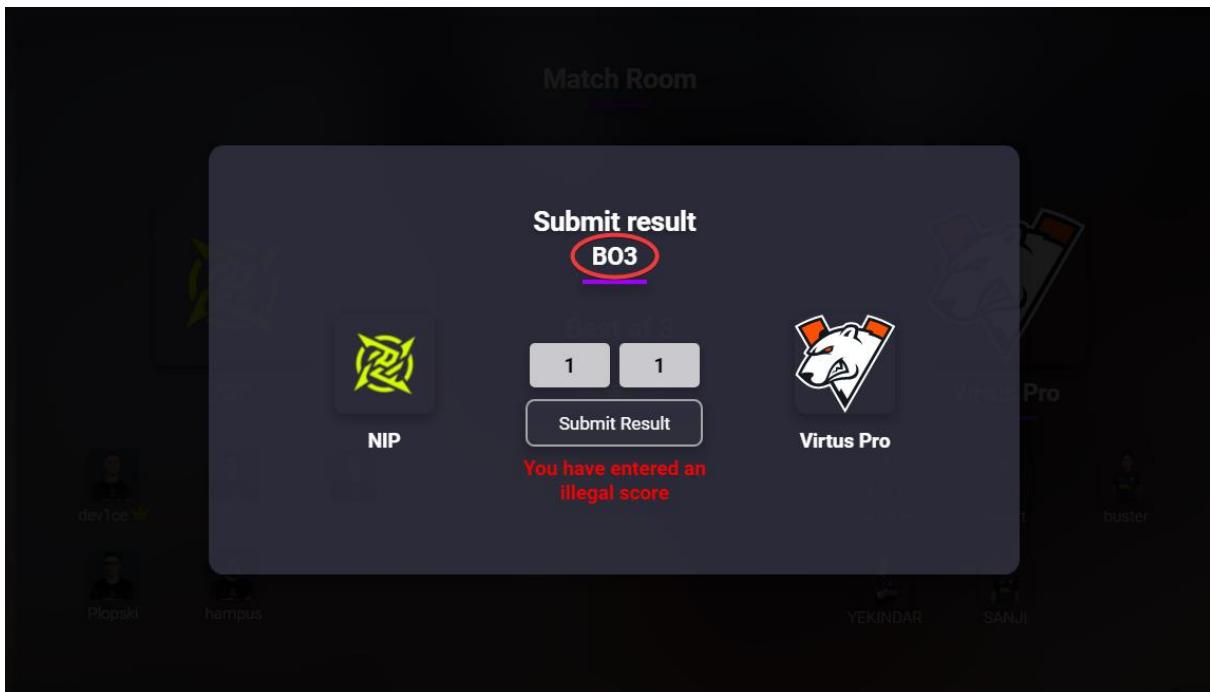
Figur 6-60 En turneringsadministrator får varsle om at en kamp skal starte



Figur 6-59 Begge lag har gjort seg klare som vist i chat, og "Ready up" knappen er blitt til en "Submit result" knapp

Når man trykker på «Submit result» åpnes et modal-vindu hvor man kan skrive inn kampresultat. Som vist i

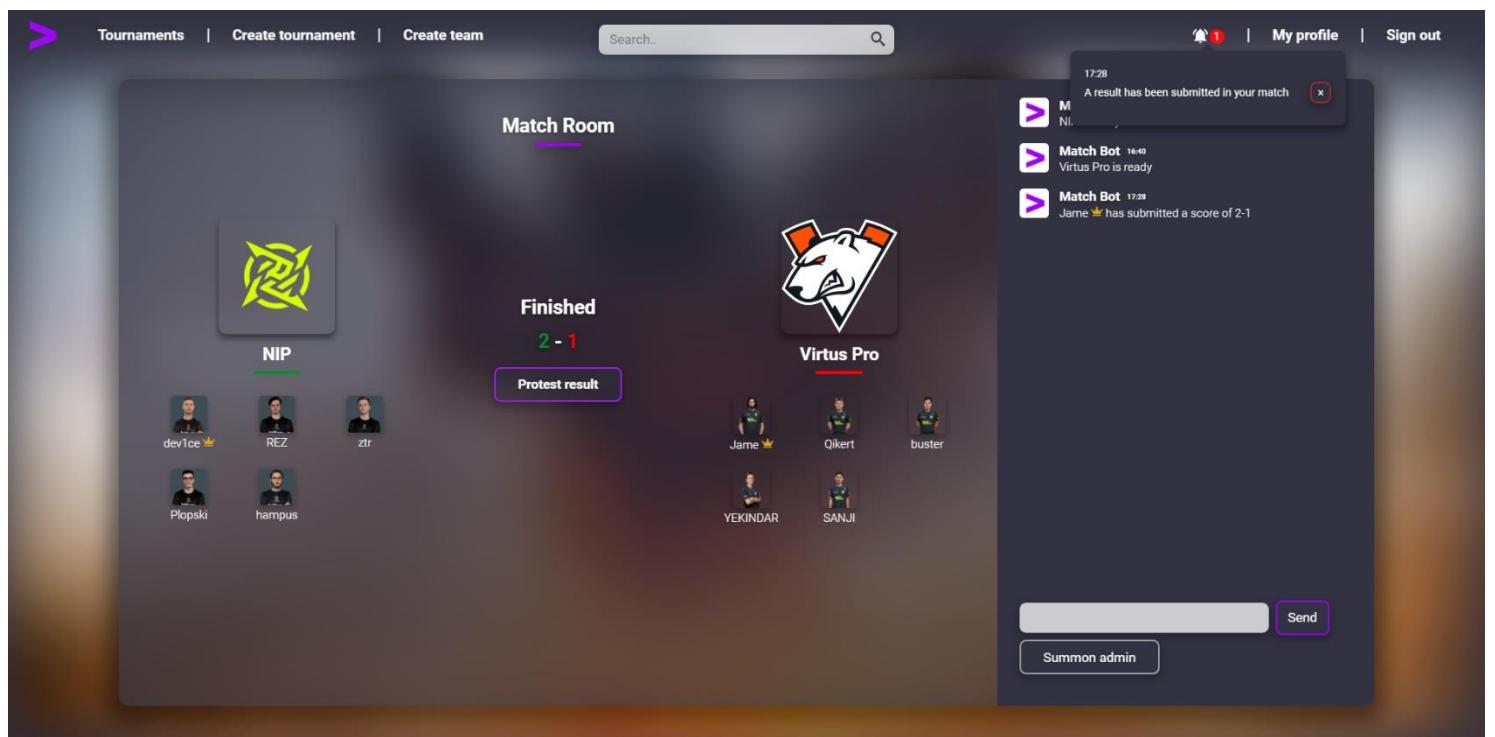
Figur 6-61 var kampen en «BO3», altså en best-av-tre. Dette betyr at laglederen bare får lov til å melde inn et resultat som er 2-0, eller 2-1.



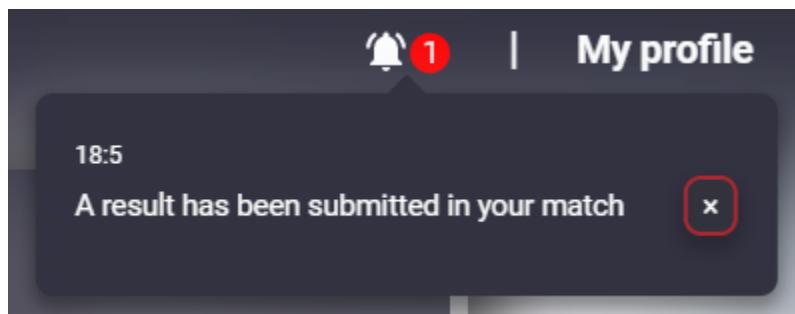
Figur 6-61 Modal-vindu for å melde resultat

Etter at en av laglederne har meldt et resultat blir resultatet automatisk satt, og vinneren blir sendt til neste kamp i turneringstreet. Alle medlemmer i kampprommet får et varsel som sier at et resultat har blitt meldt i en kamp de er med i. Det blir også sendt en melding i chatteren som sier hvem som meldte et resultat, og hva resultatet ble. Dette vises i Figur 6-62.

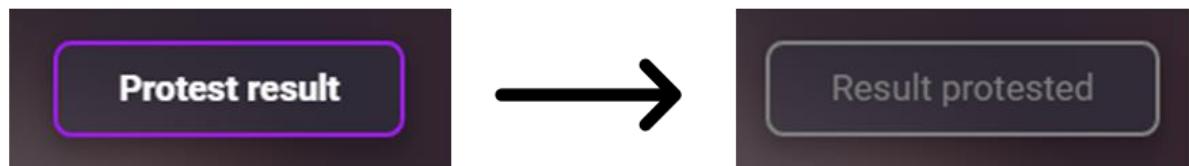
Om en lagleder er uenig i resultatet som ble meldt kan han/hun trykke på «Protest result». Da får turneringsadministrator et varsel om at et resultat har blitt protestert. Det blir også sendt en melding i chat som viser hvem som har protestert resultatet. Noe av dette vises i Figur 6-64 og Figur 6-65.



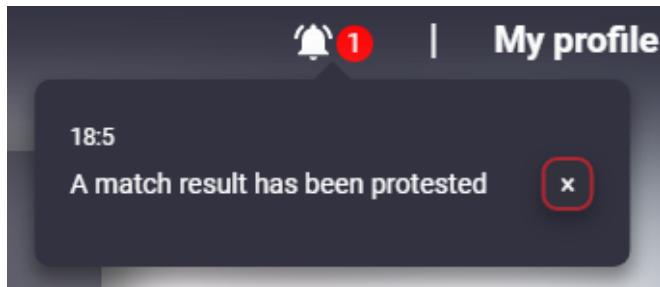
Figur 6-62 Kampdeltaker har fått varsel om at et resultat har blitt levert



Figur 6-63 Detaljer forrige bilde



Figur 6-64 Resultat har blitt protestert

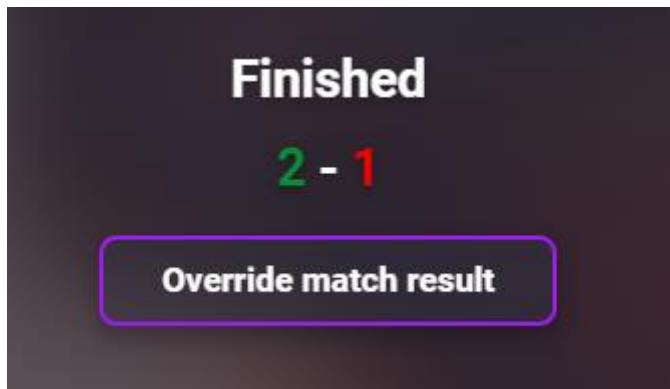


Figur 6-65 Turneringsadministrator får informasjonsvarsel om at et resultat har blitt protestert

### 6.3.10.5 Kampron for turneringsadministrator

En turneringsadministrator har muligheten til å manuelt overskrive resultatet til en kamp, uansett hvilken status en kamp har. Dette er nyttig om et lag ikke møter opp til kamp, eller om et resultat har blitt meldt feil. Å overskrive kampresultatet blir gjort gjennom en knapp alle turneringsadministratører har i kampronmet. På denne står det «Override match result». Se Figur 6-66. Når denne blir trykket på åpnes det samme modal-vinduet som blir brukt for å melde et resultat av en vanlig bruker. Se

Figur 6-61. Som nevnt tidligere har også turneringsadministrator tilgang til kampron-chatten.



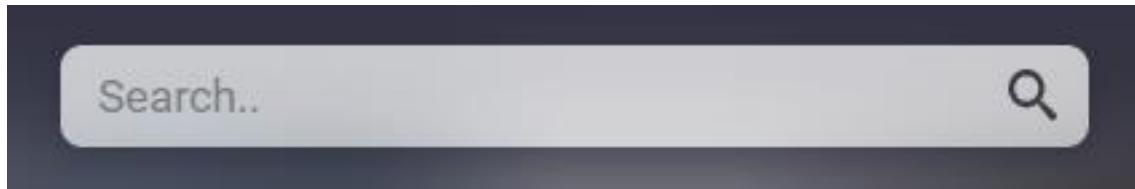
Figur 6-66 Knappen en turneringsadministrator bruker for å overskrive et resultat

### 6.3.10.6 Refleksjon over kravspesifikasjonen

Den originale kravspesifikasjonen sier at «Alle kamper i en turnering skal ha et kamprom der det står hvilke lag som møtes. Her skal det være en privat chat for alle deltagere i kampen og en administrator for tildeling av viktig informasjon knyttet til kampen.» Dette kan tolkes som at alle kamprom bare skulle ha én administrator med ansvar. Etter å ha pratet med produkteier ble det klart det ikke hadde noe å si, så lenge en deltaker lett kan kontakte turneringsadministrator, og en turneringsadministrator lett kan flytte seg til et kamprom som trenger hjelp. Produkteier er fornøyd med måten vi har løst dette på.

### 6.3.11 Søkefunksjonalitet

En bruker kan søke opp andre brukere, lag og turneringer som eksisterer på plattformen. For å benytte seg av denne funksjonaliteten skriver brukeren inn en søketekst i inputfeltet som alltid befinner seg i midten øverst på nettsiden. Videre trykker den på enten Enter-tasten eller forstørrelsesglasset som befinner seg på høyre side i inputfeltet. Når denne handlingen er utført blir brukeren presentert en utlisting av søkeresultatene i tre kolonner. Hvert av resultatene vil, ved trykk, videresende brukeren til den aktuelle profilsiden, lagsiden eller turneringssiden basert på hvilket resultat det er snakk om. Den originale kravspesifikasjonen sier at «Brukere skal kunne søke opp andre brukere, lag og turneringer.» Disse kravene er oppfylt.



Figur 6-67 Inputfeltet for søk

Tournaments | Create tournament | Create team

sherman

Search Result

Users

sherman

sherman1974

shermannen

Teams

Team Sherman

The Shermanators

Tournaments

ShermanLan 2021

Figur 6-68 Eksempel på søkeresultat med "sherman" som søketekst

### 6.3.12 Systemadministrator

Etter et møte med produkteier og flere medlemmer av Accenture sin spillgruppe, før vi skulle starte å jobbe med systemadministrator-funksjonaliteten, kom vi frem til at deres aller høyeste prioriterte ønsker for funksjonalitet til systemadministrator var å ha tabellvisning av alle brukere, lag og turneringer, og ha muligheten til å slette hver av dem. Dette møtet ble satt opp da punktene om administrering av nettstedet i kravspesifikasjonen var vag, og produkteier ønsket å få meningene til flere av brukerne som kommer til å driftet systemet.

Dette møtet skjedde rett før siste sprint. På grunn av lite tid, måtte gruppen prioritere bort noe funksjonalitet, i favør av annen. Sletting av lag ville tatt lengre tid enn forventet, da dette hadde uforutsette implikasjoner, som visning av laget i turneringstrær. Mer om disse implikasjonene i kapittel 5.6.7. Etter samtale med produkteier kom vi frem til at dette ikke var kritisk, da denne funksjonaliteten hovedsakelig skulle bli brukt for å fjerne lag under omstendigheter som at spillere hadde jukset. Istedentfor å fjerne laget kan en systemadministrator fjerne alle spillerne involvert med laget. Dette oppnår samme effekt, selv om det er mer jobb for en systemadministrator. Dermed prioriterte vi bort denne funksjonaliteten, i favør av viktigere funksjonalitet. Sletting av turneringer og brukere er implementert. Adminsidene vises i Figur 6-69, Figur 6-70 og Figur 6-71.

Username	Email	Register Date
Bubzkl	test9@email.com	22 May, 2021
Jame	test19@email.com	22 May, 2021
Kyojin	test23@email.com	22 May, 2021
Magisk	test10@email.com	22 May, 2021
Plopski	test4@email.com	22 May, 2021
Qikert	test20@email.com	22 May, 2021
REZ	test2@email.com	22 May, 2021

Figur 6-69 Utlisting av brukere med sletting implementert

Tournaments			
<input type="text" value="Search..."/>			
10 tournaments in the database			
Title	Start time	Game	
Accenture VM 2021	2021-05-26T17:30:00	Counter-Strike: Global Offensive	<button>Delete</button>
American League	2021-05-27T17:30:00	League of Legends	<button>Delete</button>
MorroCompon	2021-07-25T17:30:00	FIFA	<button>Delete</button>
LCS Proving Grounds	2021-08-25T17:30:00	Rainbow Six Siege	<button>Delete</button>
Snow Sweet Snow Weekend	2021-09-25T17:30:00	Rocket League	<button>Delete</button>
HyggeLigaen	2021-10-25T17:30:00	Valorant	<button>Delete</button>
Accenture League	2021-11-25T17:30:00	Other games	<button>Delete</button>

Figur 6-71 Utlisting av alle turneringer, med sletting implementert

Teams			
<input type="text" value="Search..."/>			
23 teams in the database			
Team name	Establish Date		
11	22 May, 2021	<button>Delete</button>	
22	22 May, 2021	<button>Delete</button>	
33	22 May, 2021	<button>Delete</button>	
44	22 May, 2021	<button>Delete</button>	
55	22 May, 2021	<button>Delete</button>	
66	22 May, 2021	<button>Delete</button>	
77	22 May, 2021	<button>Delete</button>	

Figur 6-70 Utlisting av alle lag uten sletting implementert

## **6.3.13 Andre krav fra kravspesifikasjonen**

### **6.3.13.1 Autentisering og sikkerhet**

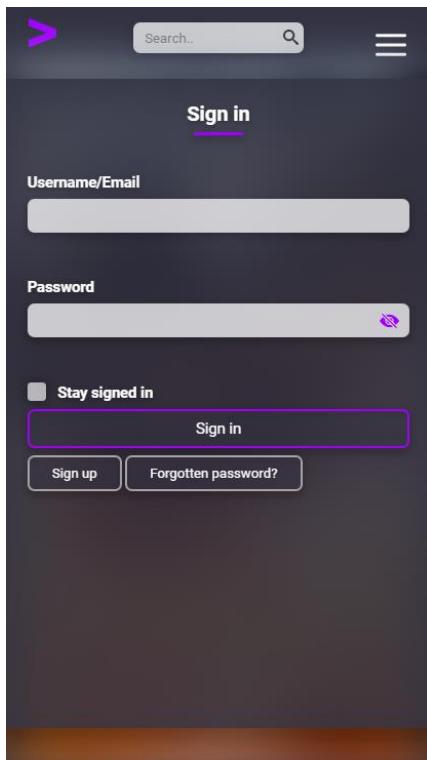
Om autentisering og sikkerhet sier kravspesifikasjonen at: «Sessions for å sjekke om bruker er innlogget og for å validere brukeren sin rolle i systemet. Informasjonskapsler (Cookies) for å holde brukeren innlogget». Disse kravene er innfridd og blir dokumentert i kapittel 6.4.6.5.

### **6.3.13.2 Universell utforming**

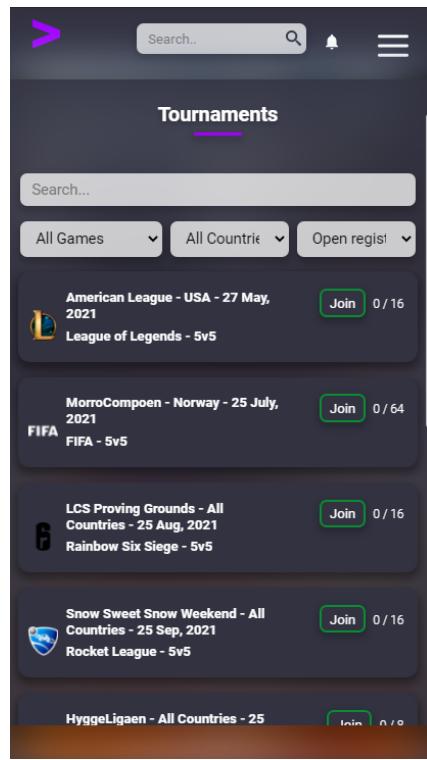
Kravspesifikasjonen sier at «Nettløsninger skal minst utformes i samsvar med standard Web Content Accessibility Guidelines 2.0 (WCAG 2.0)/NS/ISO/IEC 40500:2012, på nivå A og AA med unntak for suksesskriteriene 1.2.3, 1.2.4 og 1.2.5, eller tilsvarende denne standard. Jf. § 4 i Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger». Disse kravene er innfridd og blir dokumentert i delkapittel 7.5.3.

### **6.3.13.3 Responsivt design**

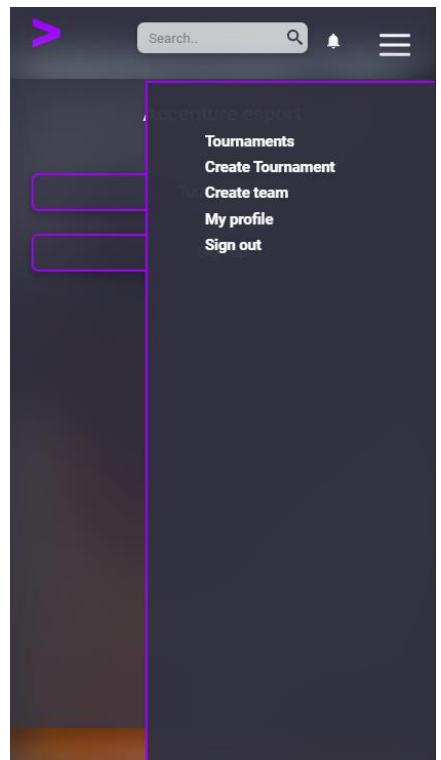
Kravspesifikasjonen sier at «Webapplikasjonen skal skalere til mindre skjermer». Dette har ikke vært høyeste prioritering, da webapplikasjonen hovedsakelig skal brukes på datamaskiner. Målet var at det er mulig å bruke nettsiden på så små skjermer som mobiler. Dette var med på å styre designet vårt, som det gjennomgående designvalget med å ha en boks med innhold midt på siden. Dette er fordi denne boksen er enkel å skalere til forskjellige skermstørrelser. I tillegg har vi implementert en hamburger-meny som tar over for navbaren for mindre skjermer. Flere utklipp av webapplikasjonen i mobilstørrelse ligger vedlagt under. Disse ble lagd gjennom «chrome devtools» sin funksjonalitet for skalering av nettsider til mobilstørrelse. Disse skjermbildene er skalert til en Iphone 6/7/8 plus.



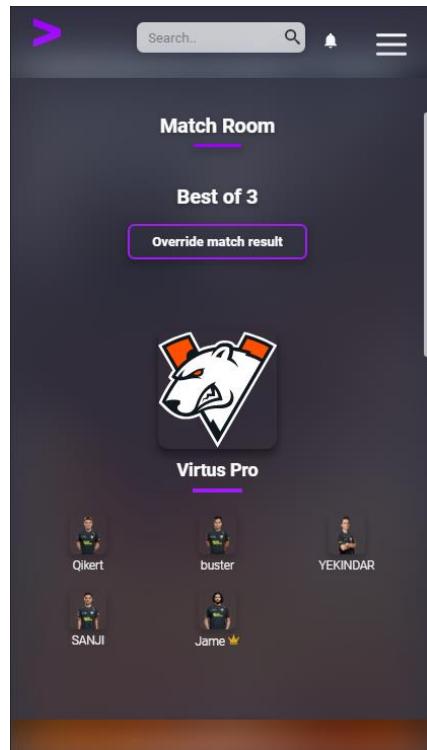
Figur 6-75: Sign in på mobil



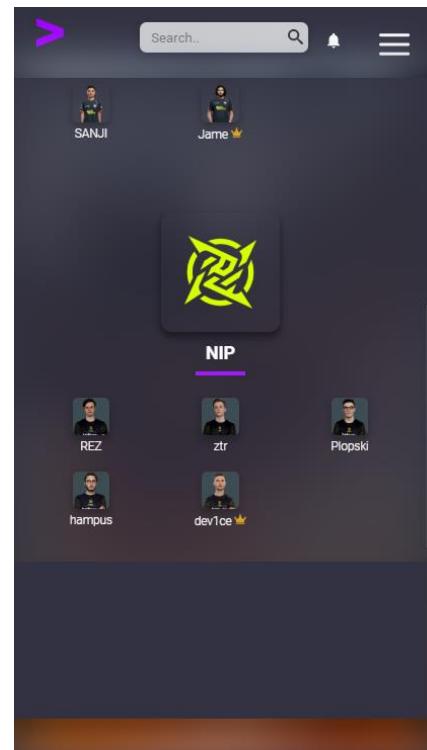
Figur 6-76: Turneringsutlisting på mobil



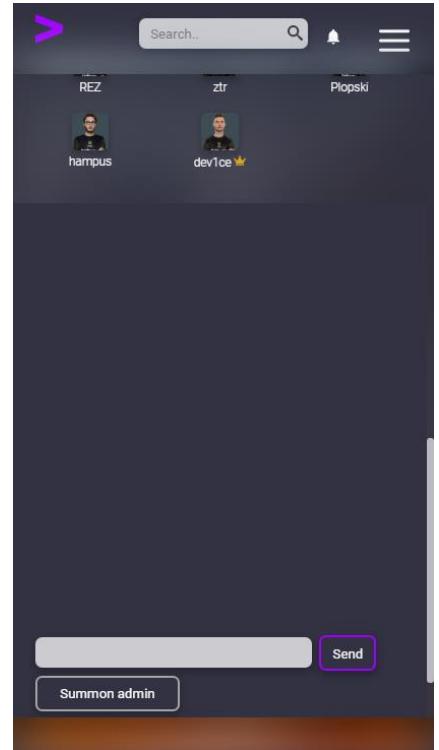
Figur 6-77: Hamburgermeny på mobil



Figur 6-72: Kamprom på mobil del 1



Figur 6-73: Kamprom på mobil del 2



Figur 6-74: Kamprom på mobil del 3

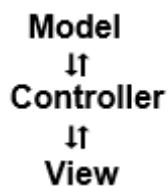
#### **6.3.13.4 Rammebetingelser**

Prosjektets rammebetingelser sier at løsningen skal utvikles som en webapplikasjon, gruppen skal bruke en smidig prosessmodell, språket i brukergrensesnittet skal være norsk, og at gruppen skal bruke teknologiene React og .NET. Prosessen vår er dokumentert i prosessdokumentasjonen. Underveis i prosjektet oppdaterte rammebetingelsene seg, ved at språket på brukergrensesnittet skulle gjøres om til engelsk. Dette har vi gjort. Gruppen har brukt teknologiene React og .NET. Rammebetingelsene er innfridd.

## 6.4 Programmets oppbygging og virkemåte

### 6.4.1 Systemarkitektur

Arkitektur til systemet vårt er bygget på designmønsteret Model–View–Controller (MVC). Dette er en lagdelt arkitektur som deler systemet vårt i tre lag: Model, View og Controller. Hvert lag kan bare prate med laget over, eller under seg. Dette er illustrert i Figur 6-78. Model er den delen som tar seg av datahåndtering. For eksempel så vil et brukerobjekt hente data fra databasen, oppdatere dataene og lagre de til databasen. View er den delen som håndterer brukergrensesnittet. Controller er den delen som fungerer som et bindeledd mellom Model og View, som håndterer feil slik at belastningen på Model og View blir mindre. (Tutorialspoint, 2021)



Figur 6-78 MVC

### 6.4.2 Serverside

#### 6.4.2.1 Installerte avhengigheter

Vi benyttet NuGet sitt installasjonsverktøy i Visual Studio, for å installere de nødvendige avhengighetene på serversiden<sup>2</sup>.

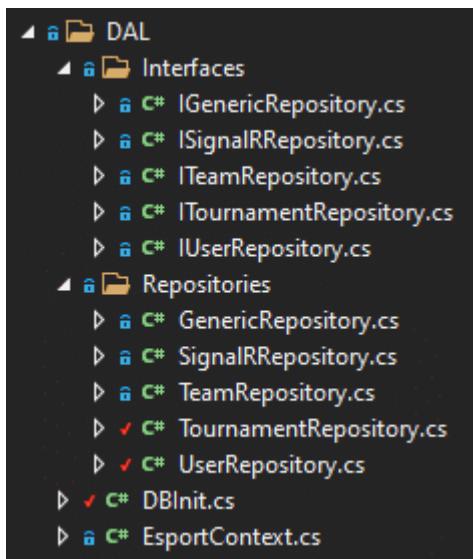
Installerte avhengigheter	Bruksområde

<sup>2</sup> Se Ordliste kapittel 9

AspNet.Security.OpenId.Steam (3.1.0)	Kommunisere med steam sitt API
MailKit (2.0.2)	Sende Mail
Microsoft.AspNet.WebApi.Client (5.2.7)	Oppsett for API
Microsoft.AspNetCore.SignalR.Client (5.0.5)	Sanntid kommunikasjon mellom klienter
Microsoft.AspNetCore.SpaServices.Extensions (3.1.10)	Oppsett for Single Page Application
Microsoft.EntityFrameworkCore (5.0.2)	Abstraksjon av databasen
Microsoft.EntityFrameworkCore.Proxies (5.0.2)	Abstraksjon av databasen, nødvendig ved bruk av lazyloading
Microsoft.EntityFrameworkCore.Sqlite (5.0.2)	Abstraksjon av SQLite database
Microsoft.EntityFrameworkCore.SqlServer (3.1.3)	Abstraksjon av SQLServer database
Microsoft.EntityFrameworkCore.Tools (3.1.3)	Abstraksjon av databasen, nødvendig ved bruk av SQLServer og ConsmosDB
Serilog.Extensions.Logging.File (2.0.0)	Logging til fil

#### 6.4.2.2 Data Access Layer

Systemet vårt har en egen mappe for databasehåndtering kalt «Data Access-Layer» (DAL). Her ligger all databaserelatert kode. Et utklipp av dette vises i Figur 6-79. Et repository inneholder kode brukt for å hente ut, lagre eller slette verdier i databasen. UserRepository henter for eksempel ut informasjon relatert til en User-tabell i databasen. I DALen opprettes også databasen ved hjelp av Code-First tilnærmingen med Entity Framework Core (se neste avsnitt). DALen representerer Model i MVC.



Figur 6-79 Database access layer

#### 6.4.2.3 Code-First Database

Vi opprettet databasen med en “Code-First”-tilnærming. Ved hjelp av rammeverket «Entity Framework Core» kunne vi opprette databasestrukturen vår gjennom C#-klasser, og så ble det automatisk gjort noe normalisering, ved at for eksempel lister ble transformert til hjelpetabeller, for mer effektiv lagring. Løsningen vår inneholder en SQL-databasen, men vi snakker ikke direkte med databasen gjennom SQL-spørninger. Entity Framework Core fungerer som et abstraksjonslag mellom koden vår og databasen. Som erstatning for SQL-spørninger benytter vi «Language-Integrated Query» (LINQ). Ved hjelp av LINQ kunne vi aksessere og manipulere lister med objekter av C#-klassene som vi brukte for å opprette databasen direkte, framfor å jobbe

mot den normaliserte strukturen som ligger lagret i databasen. En annet fordel med LINQ er at det også kan brukes for å manipulere data i resten av programmet, noe som bidro til effektivitet i utviklingen.

#### 6.4.2.4 Controller

Controller-laget i applikasjonen kobler sammen klientsiden og DALen. En controller er full av API-endepunkter som kan bli kalt fra klientkoden gjennom http-forespørslar av typen get og post.

For eksempel, om en klient ønsker å registrere en bruker på applikasjonen vil den fylle inn nødvendig informasjon på klientsiden og utføre en post-forespørsel. Deretter kommer controlleren til å til å validere om all informasjonen som ble sendt med er gyldig, og om klienten som kalte API-endepunktet, om nødvendig, har rettighetene til dette. Deretter sendes denne informasjonen videre til DALen. DALen legger informasjonen til den nye brukeren inn i databasen, og brukeren er registrert.

På klientsiden har vi valgt å bruke JavaScript-biblioteket Axios for å sende http-forespørslar. I Figur 6-80 og Figur 6-81 viser vi hvordan et API-endepunkt ser ut i en controller, og hvordan man kaller dette fra klientsiden. `[HttpPost("")]` viser at man kaller API-endepunktet med en post-forespørsel. API-endepunktet heter «SendMessageToGroup», og ligger i SignalRController. Man vet dermed at man må sende en post-forespørsel til «/SignalR/SendMessageToGroup».

```
[HttpPost("SendMessageToGroup")]
public async Task<ActionResult> SendMessageToGroup(ChatMessageModel message)
```

Figur 6-80: API-endepunkt i controller

```
await axios.post("/SignalR/SendMessageToGroup", chatMessage);
```

Figur 6-81: Bruker Axios til å utføre en postforespørsel

#### 6.4.2.5 Models

Systemets «Models»-mappe må ikke blandes med utrykket Model i MVC arkitekturen. Models mappe inneholder modell-objekter som blir brukt til kommunikasjon mellom serversiden og klientsiden. En måte dette blir gjort på vises i Figur 6-83. På klientsiden blir det opprettet et JavaScript-objekt: «chatMessage». Dette blir sendt med i en http-forespørsel til API-endepunktet «/SignalR/SendMessageToGroup». Figur 6-84 viser at denne controller-metoden har input-parameter av typen «ChatMessageModel». Om man sammenligner «chatMessage» fra Figur 6-83 og «ChatMessageModel» i Figur 6-82, er de helt like. Dermed konverteres JavaScript-objektet som blir sendt til API-endepunktet til en «ChatMessageModel», og det blir brukt som input parameter i controller-metoden. Hvis JavaScriptobjektet brukt på klientsiden, og modell-objektet brukt på serversiden matcher, er det altså mulig å sende objekter frem og tilbake fra serverside til klientside.



```
handleSendMessage = async (message) => {
  if (this.props.connection.connectionStarted) {
    const { user } = this.state;
    const date = new Date();
    const time = `${date.getHours()}:${date.getMinutes()}`;
    const chatMessage = {
      group: this.props.match.params.id,
      userId: "",
      username: user.username,
      profileImagePath: user.profileImagePath,
      message: message,
      time: time,
    };
    try {
      await axios.post("/SignalR/SendMessageToGroup", chatMessage);
    } catch (e) {
      console.log(e);
    }
  } else {
    alert("No connection to server yet.");
  }
};
```

```
namespace Accenture_e_sportportal.Models
{
  public class ChatMessageModel
  {
    public string Group { get; set; }
    public string UserId { get; set; }
    public string Username { get; set; }
    public string ProfileImagePath { get; set; }
    public string Message { get; set; }
    public string Time { get; set; }
  }
}
```

Figur 6-82 Modell-objekt som matcher JavaScript-objektet sendt inn med post-forespørsel

Figur 6-83: JavaScript-objekt opprettes på klienten og blir sendt med post-forespørsel

```
[HttpPost("SendMessageToGroup")]
public async Task<ActionResult> SendMessageToGroup(ChatMessageModel message)
```

Figur 6-84: API-endepunkt i en controller

#### 6.4.2.6 Logging

Hvis deler av koden feiler på serversiden blir feilmeldingen(e) skrevet til egne log filer. Feilmeldinger som blir logget samme dag blir skrevet til samme fil, og første feilmelding for dagen oppretter en nye loggfil.

```
    catch (Exception e)
    {
        _log.LogWarning(e.Message);
        return BadRequest("Could not get tournament");
    }
```

Figur 6-85 Logger feilmelding

#### 6.4.2.7 Tråder

Vi har benyttet oss av asynkron programmering hvor vi sørget for at databasetransaksjonene i DALen blir gjennomført i ulike tråder, der dette er hensiktsmessig. Ved å bruke dette blir koden langt mer effektiv i en del tilfeller. I figur 6-80 er et eksempel på fire kall til metoder i DALen som utføres asynkront, og deretter forsikrer man at det er gjennomfør med nøkkelordet await..

```
Task<List<Users>> usersAsync = _user.SearchForUsername(searchForUsername.SearchString);

Task<Teams> team = _team.GetTeamById(searchForUsername.TeamId);
Task<List<Users>> existingLeaders = _team.GetTeamLeaders(searchForUsername.TeamId);
Task<List<Users>> invitedUsers = _team.GetInvitedUsers(await team);
Task<List<Users>> existingMembers = _team.GetTeamMembers(searchForUsername.TeamId);

List<Users> remove = await existingLeaders;
remove = remove.Concat(await invitedUsers).ToList();
remove = remove.Concat(await existingMembers).ToList();

List<Users> users = await usersAsync;
users = users.Except(remove).ToList();
```

Figur 6-86 Eksempel på bruk av asynkron programmering

## 6.4.3 Klientside

### 6.4.3.1 React

På klientsiden har vi brukt JavaScript-biblioteket React. I forhold til rammeverker som Angular, er React ikke et fullstendig rammeverk. Derfor har ikke React alltid innebygde løsninger som for eksempel routing. Istedentfor laster man ned tredjepartsbiblioteker som gir ønsket funksjonalitet. Siden React er så populært er det mange biblioteker å velge mellom, og dette gir stor fleksibilitet. Dette ser vi på som en fordel.

React gjør det mulig å lage «Single Page Applications» (SPA). En vanlig nettside fungerer på følgende måte: Brukeren sender en http-forespørsel til nettsidens webserver, som returner nettsidens landingside. Hver gang webserveren returnerer en side til nettleseren, må denne siden bli lastet inn. Om brukeren ønsker å navigere seg til en ny side inne på nettsiden blir det gjort en ny http-forespørsel til webserveren. På denne måten må nettleseren laste inn en helt ny side, hver gang brukeren nавигerer seg rundt på nettsiden. En SPA løser dette problemet. Når man besøker en SPA laster nettleseren inn et helt JavaScript-program, som blir kjørt opp i bakgrunnen. Dette tar ikke mer enn noen sekunder, men det tar fortsatt lengre tid å laste inn, enn vanlige nettsider. Dette JavaScript-programmet inneholder all informasjon som trengs for å forandre innholdet på siden når man nавигerer seg rundt. Når den trenger å kontakte webserveren for noe informasjon, så kan den hente den spesifikke informasjonen man trenger, og ikke en helt ny side som krever at nettleseren laster inn hele siden på nytt. Dette gjør så nettsiden føles flytende ut, og gir en mye bedre brukeropplevelse, da man aldri behøver å vente på at nettleseren laster inn siden mer enn første gang man besøker den.

### 6.4.3.2 Installerte avhengigheter

Vi benyttet terminalkomandoen «npm install <navn på avhengigheten>» for å installere de nødvendige avhengighetene på klientsiden.

Installerte avhengigheter	Bruksområde
axios	Servekall
joi	Frontendvalidering
react-brackets	Base for turingstreet
react-icons	Ikoner
uuid	Generering av GUIDs
react-image-cropper	Beskjæring av bilder
react-easy-panzoom	Manøvrering og zooming i turneringstreet
@microsoft/signalr	Sanntid kommunikasjon mellom klienter
react-scroll	Automatisk scrolling til bunn av chat ved ny melding
react-router-dom	Routing i applikasjonen

#### 6.4.3.3 Komponenter

Komponenter er en sentral del av hvordan man jobber med React. En komponent er en av flere byggeklosser som utgjør et brukergrensesnitt. En React applikasjon er bygd

opp som et tre av komponenter. Se Figur 6-87 for kommende forklaring. Den gule boksen som strekker seg hele veien rundt figuren er rot-komponenten. Dette er den ytterste komponenten i treet og inneholder alle treets komponenter. Den neste grenen i treet er visualisert med de rosa boksene. Disse er egne komponenter, som har sine egne grener innover i komponenttreeet. Dette fortsetter med et par utvalgt komponenter som vises med rødt og blått. Dette er måten man kan bruke komponenter til å bygge avanserte brukergrensesnitt i React.



Figur 6-87: Visualisering av komponenter, vist med forskjellige farger

For å ha komponenter som forandrer seg dynamisk har vi valgt å bruke klassekomponenter med «state». State er data som tilhører komponenten, og som man kan oppdatere ved brukerinteraksjoner. I Figur 6-88 ligger et skjermbilde av staten til en brukerprofil. Hvis man åpner en brukerprofil, blir det automatisk gjort validering på serversiden, og om brukeren er eier av brukerprofilen blir «isOwner» satt til true. Da vises alle redigeringsknappene på en brukerprofil, som lar brukeren bytte

profilbilde, legger til spillkontoer osv. Hvis en bruker velger å oppdatere profilbildet sitt så forandres staten til «profileImagePath» til pathen til det nye profilbildet, og profilbilde oppdateres for brukeren. All funksjonalitet som blir tilgjengelig ved bytte av state valideres på serversiden, av sikkerhetsårsaker.

```
class UserProfile extends Component {
  state = {
    isOwner: false,
    bannerPath: this.props.bannerPlaceholder,
    profileImagePath: this.props.profileImagePlaceholder,
    username: "...",
    registerDate: "",
    teamList: [],
    steamId: "",
    discord: "",
    modalViews: {
      showModal: false,
      showUpdateProfileBanner: false,
      showUpdateProfileImage: false,
      showAddGameAccounts: false,
      showEditUser: false,
      showAddTeam: false,
    },
  };
}
```

Figur 6-88 state i klassekomponent

State er ikke den eneste måten å forandre utseende på en komponent. Staten til en komponent lever bare i den spesifikke komponenten, og det er ofte man ønsker at forandringer gjort i en komponent skal påvirke en annen. I dette tilfellet bruker man «props» eller «events». Props og events er måten man sender informasjon opp og ned i komponenttreer. For å sende informasjon nedover bruker man props, og for å sende informasjon oppover bruker man events. I figurnavn vises en funksjonell komponent. Denne er «stateless», noe som vil si at den ikke har noen state. Denne komponenten blir bare påvirket av props som blir sendt inn fra andre komponenter høyre i komponentreet. ProfileImagePath er altså en prop sendt fra en komponent høyre i komponentreet som påvirker utseende til denne komponenten.

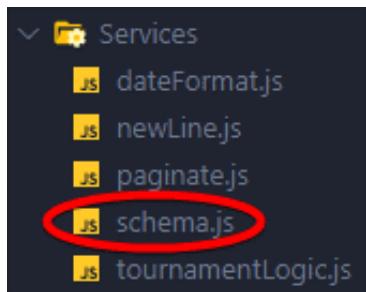
```
const Message = ({  
  profileImagePath,  
  message,  
  time,  
  username,  
  onAddRoleIcon,  
}) => (  
  <div className="message">  
    <img className="sender-image" src={profileImagePath} alt="Sender" />  
  
    <h1>  
      <span>{onAddRoleIcon(username)}</span>  
      <small>{time}</small>  
    </h1>  
    <p>{onAddRoleIcon(message)}</p>  
  </div>  
);  
  
export default Message;
```

Figur 6-89 props i funksjonell komponent

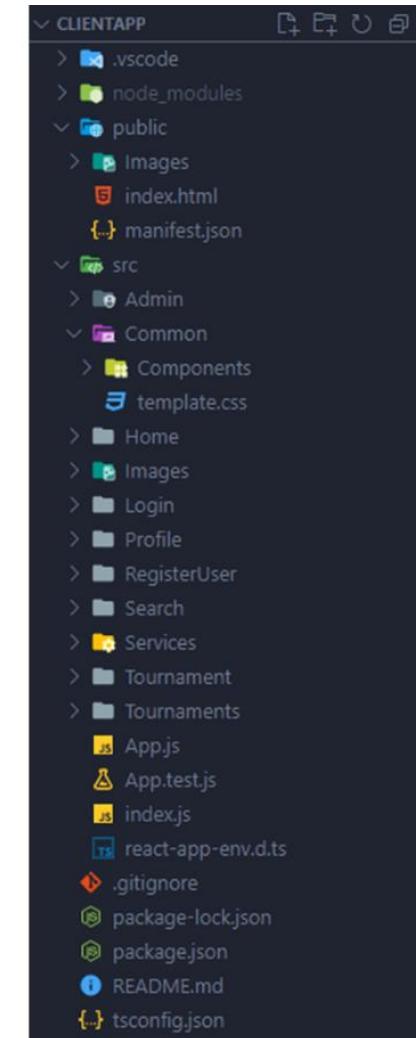
#### 6.4.3.4 ClientApp

Klientkoden ligger i en mappe som heter «ClientApp». Her ligger alle komponentene i prosjektet, og JavaScript-kode som utgjør brukergrensesnittet. Gjenbruksbare-komponenter som ikke tilhører noe spesifikt sted, blir lagt i en «Common»-mappe. Resten av mappene utenom «Services» og «Images» inneholder komponenter som bare blir brukt på sitt område i applikasjonen. For eksempel inneholder «Tournaments»-mappen alle de ikke-gjenbruksbare-komponentene som ble brukt for å vise utlistingen av alle turneringer i Figur 6-30.

Services-mappen inneholder forskjellige JavaScript-filer med logikk som blir brukt flere steder i prosjektet. Som vist i Figur 6-91 inneholder Services-mappen en fil som heter «Schema». Denne inneholder all valideringslogikk for klientsiden. ClientApp representerer View i MVC.



Figur 6-91 Schema.js



Figur 6-90 ClientApp mappestruktur

#### 6.4.3.5 Form-klassen og Input-komponenten

Form er en klasse som inneholder logikk som er felles for alle skjemaer i webapplikasjonen. Her finner vi blant annet kode for validering av hvert enkelt inputfelt, validering av alle input-felt og innsending av skjema. En klasse-komponent som inneholder et skjema blir satt til å arve fra form-klassen, og får da tilgang til all koden

den inkluderer. Input-komponenten blir benyttet for alle inputfeltet i et skjema. Komponenten er veldig fleksibel ved at det, i tillegg til et vanlig inputfelt, er mulig å formtere den som passordfelt eller tekstområde. Dette med å kun gjøre små forandringer i props verdiene man sender til denne komponenten. Vi innså før utviklingen startet at webapplikasjonen kom til å inneholde en god del skjemaer. Ved å bruke litt ekstra tid for å sette opp form-klassen og input-komponenten i starten av utviklingen, sparte vi oss for store mengder arbeid gjennom prosjektet.

## 6.4.4 Turneringslogikk

Det er store deler programlogikk involvert gjennom livssyklus til en turnering. I denne delen presenteres først de sentrale datastrukturene for en turnering. Videre blir det redegjort for turneringslogikken og vil bli gått dypere inn på de mest sentrale delene. Alle operasjoner som er redegjort i denne delen inkluderer kode for å forsikre at brukeren er innlogget for å kunne gjennomføre de ulike operasjonene. Dette blir derfor ikke nevnt spesifikt for hver av dem. Hver av operasjonene har et ansvarlig endepunkt, og figur med identifikasjon til hver av disse ligger før redegjørelsen. Referanser til en turnerings statuskode er en gjenganger i denne delen. En fullstendig oversikt over disse statuskodene er å finne i Brukermanual.

### 6.4.4.1 Datastruktur

Selv om vi visualiserer kampoversikten som et turneringstre (nodetre), så benytter vi oss ikke av denne datastrukturen for å lagre eller bearbeide kampene i turneringen. Vi vurderte det som langt mer praktisk å benytte oss av en matrise av Match-objekter både med tanke på lagring og bearbeiding.

```
[0][1][2][3][4][5][6][7]
 [0]  [1]  [2]  [3]
   [0]      [1]
     [0]
```

Figur 6-92: Matrise med kampobjekter,

#### 6.4.4.2 Opprettelse

```
[HttpPost("NewTournament")]
public async Task<ActionResult> NewTournament(NewTournamentModel newTournament)
```

Figur 6-93 Identifikasjon for NewTournament-endepunkt

Det første steget i en turnering sin livssyklus er opprettelse. Etter at en bruker har formatert turneringen etter ønske i brukergrensesnittet, blir informasjonen sendt til serveren gjennom en post-forespørsel til endepunktet «/Tournament/NewTournament». Det blir opprettet et nytt Tournament-objekt som blir lagt til i databasen. Videre blir det opprettet en liste av TournamentAdmin-objekter som inneholder User-objektene til brukerne som skal fungere som administrator for turneringen. Denne listen blir lagt til i Tournament-objektet. Alle bruker som er lagt til i denne listen blir referert til som turnerings-admin videre i rapporten. Turneringsstatus blir satt til 0 etter opprettelse.

#### 6.4.4.3 Påmelding

```
[HttpPost("Join")]
public async Task<ActionResult> Join(TournamentAndTeamIdModel inputModel)
```

Figur 6-94 Identifikasjon for Join-endepunkt

Ved status 0 er turneringen i påmeldingsfasen av sin livssyklus. I denne fasen er det mulig for alle lagledere å melde laget sitt på en turnering. Etter at laglederen har trykket «Join»-knappen til en turnering, i brukergrensesnittet, blir IDen til både turneringen og laget lagt til i et objekt som sendes til serveren gjennom en post-forespørsel til endepunktet «/Tournament/Join». Her blir det sjekket at brukeren er lagleder for laget den prøver å melde på, og om turneringsstatus er like 0.

#### 6.4.4.4 Lukke påmelding

```
[HttpGet("CloseTournament")]
public async Task<ActionResult> CloseTournament(string tournamentId)
```

Figur 6-95 Identifikasjon for CloseTournament-endepunkt

Turneringsadministratoren(e) kan når som helst lukke en turnering gitt at status er lik 0 og det er minimum 4 lag påmeldt. Dette gjøres med en get-forespørsel til endepunktet «/Tournament/CloseTournament» med turneringsID som parameter. Dette endepunktet iverksetter en rekke operasjon. Figuren under viser de fem kodelinjene vi vider skal se nøyere på.

```
List<Match> matches = TournamentLogic.GenerateTournamentBlueprint(tournament);
await _tournament.AddTournamentBlueprint(matches);
await _tournament.ScrambleFirstRound(tournament);
await _tournament.HandleWalkover(tournament);
await _tournament.ChangeTournamentStatus(1, tournamentId);
```

Figur 6-96 Operasjonene vi skal se nøyere på

##### 6.4.4.4.1 Kodelinje 1

Første kodelinje er å benytte metoden «GenerateTournamentBlueprint» for å generere en liste med Match-objekter. Først blir metodene «SpotsInFirstRound» og «NumberOfRound», med antall påmeldte lag som argument, brukt for å bestemme størrelsen på turneringstreet. Deretter blir det benyttet to nestede for-løkker, hvor den ytterste itererer basert på antall runder og den innerste iterer basert på antall kamper i den aktuelle runden.

```

public static List<Match> GenerateTournamentBlueprint(Tournament tournament)
{
    List<Match> matches = new List<Match>();

    int teamsCount = tournament.Teams.Count;
    int spotsInFirstRound = SpotsInFirstRound(teamsCount);
    int roundsCount = NumberOfRounds(teamsCount);
    int currentRoundMatchCount = spotsInFirstRound / 2;
    int currentRoundTeamsCount = spotsInFirstRound;

    // Looping through every round in the tournament
    for (int i = 0; i < roundsCount; i++)
    {
        int currentRoundFormat = RoundFormat(tournament, currentRoundTeamsCount);

        // Looping through every match in current round
        for (int j = 0; j < currentRoundMatchCount; j++)
        {
            Match match = new Match()
            {
                Id = Guid.NewGuid(),
                Round = i,
                RoundIndex = j,
                TeamsThisRound = currentRoundTeamsCount,
                BestOf = currentRoundFormat,
                Tournament = tournament
            };
            matches.Add(match);
        }

        // Dividing teams and round count by 2 before next round
        currentRoundTeamsCount /= 2;
        currentRoundMatchCount /= 2;
    }

    return matches;
}

```

Figur 6-97 metode generateTournamentBlueprint

```

private static int SpotsInFirstRound(int numberOfTeams)
{
    return (int)Math.Pow(2, Math.Ceiling(Math.Log(numberOfTeams, 2)));
}

```

Figur 6-98 metoden SpotsInFirstRound

```
public static int NumberOfRounds(int numberOfTeams)
{
    return (int)Math.Ceiling(Math.Log(numberOfTeams, 2));
}
```

Figur 6-99 metoden *numberOfRounds*

Hvert Match-objekt blir opprettet med heltallsverdiene Round og RoundsIndex. Round definerer hvilken runde den aktuelle kampen skal spilles i, og RoundIndex definerer hvilken posisjon denne kampen har i den aktuelle runden. På figuren under er matrisen med Match-objekter visualisert for en turnering med 8 kamper i første runde.



Figur 4-6-100 Rundeindeks visualisering

#### 6.4.4.4.2 Kodelinje 2

Kodelinje to benytter DAL-metoden «AddTournamentBlueprint» for å legge alle Match-objektene til i databasen.

```
public async Task AddTournamentBlueprint(List<Match> matches)
{
    _db.Matches.AddRange(matches);
    await _db.SaveChangesAsync();
}
```

Figur 6-101 metoden «AddTournamentBlueprint»

#### 6.4.4.4.3 Kodelinje 3

Kodelinje tre benytter DAL-metoden «ScrambleFirstRound» for å stokke listen med påmeldte lag og deretter plassere de inn i de allerede opprettede Match-objektene hvor *Round* = 0. Deretter lagres endringene i databasen.

```
public async Task ScrambleFirstRound(Tournament tournament)
{
    List<Match> firstRoundMatches = await GetFirstRoundMatches(tournament);
    TournamentLogic.ScrambleFirstRound(firstRoundMatches, tournament);
    _db.SaveChanges();
}
```

Figur 6-102 metoden «ScrambleFirstRound»

Plasseringen av lag i første runde blir gjort etter et bestemt mønster. Alle lagene som har partallindeks i den stokkede listen blir plassert som Team1 og alle lagene som har oddetallindeks blir plassert som Team2 når de plasseres i et Match-objekt. Lagene med partallindeks itererer gjennom listen med kamper i første runde fra Index 0 og oppover, mens lagene med oddetallindeks itererer gjennom listen fra siste indeks og nedover. Denne algoritmen gir mulighet for å implementere rangering av lagene i fremtiden, slik at det antatt best laget møter det dårligste, det nest beste møter det nest dårligste osv. Algoritmen vil også sørge for at turneringstreet blir så balansert som

mulig i tilfeller hvor det ikke er nok lag til å fylle alle kampene i første runde. Dette fordi kampene med kun et lag, i praksis vil jobbe seg gradvis mot midten fra hver sin side av turneringstreet. På figurene Figur 6-103, Figur 6-104 og Figur 6-105, ser vi hvordan fordeling ser ut for en turnering med henholdsvis 8, 6 og 5 påmeldte lag. Figur 6-106 viser algoritmen som er omtalt i dette avsnittet.

[0][7] [2][5] [4][3] [6][1]

Figur 6-103 Turnering med 8 lag

[0][ ] [2][5] [4][3] [ ][1]

Figur 6-104 Turnering med 6 lag

[0][ ] [2][ ] [4][3] [ ][1]

Figur 6-105 Turnering med 5 lag

```

public static void ScrambleFirstRound(List<Match> firstRoundMatches, Tournament tournament)
{
    tournament.Teams.Shuffle(); // Shuffling the list of teams to create random matchups

    int teamsCount = tournament.Teams.Count;
    int spotsInFirstRound = firstRoundMatches.Count * 2;

    int currentTeam1Spot = 0; // First index in the list of matches
    int currentTeam2Spot = firstRoundMatches.Count - 1; // Last index in the list of matches
    int i = 0;

    // Looping through all spots in the first round
    while (i < spotsInFirstRound)
    {
        if (i % 2 == 0) // Even number
        {
            if (i < teamsCount)
            {
                firstRoundMatches[currentTeam1Spot].Team1 = tournament.Teams[i];
            }
            else
            {
                // Empty team in case the list of teams are not enough to fill all spots in the first round
                firstRoundMatches[currentTeam1Spot].Team1 = new TournamentTeam();
            }
            currentTeam1Spot++;
        }
        else // Odd number
        {
            if (i < teamsCount)
            {
                firstRoundMatches[currentTeam2Spot].Team2 = tournament.Teams[i];
            }
            else
            {
                // Empty team in case the list of teams are not enough to fill all spots in the first round
                firstRoundMatches[currentTeam2Spot].Team2 = new TournamentTeam();
            }
            currentTeam2Spot--;
        }
        i++;
    }
}

```

Figur 6-106 Algoritme for å fordele lag i første runde

#### 6.4.4.4 Kodelinje 4

Kodelinje 4 tar for seg tilfellet hvor det ikke er nok påmeldte lag for å fylle alle kampene i første runde. DAL-metoden “HandleWalkover” itererer gjennom alle kampene i første runde og sjekker for hver av dem om enten Team1 eller Team2 mangler en referanse til et lag. I tilfelle hvor dette gjelder Team1, så blir Team2 satt som vinner av kampen, og flyttet til neste runde. Tilsvarende i motsatt tilfelle for Team2. Hvordan algoritmen for å deklarerer og flytte en vinner til neste runde fungere, blir redegjort for i delen som omhandler melding av resultat, se kapittel 6.4.4.6.

```

public async Task HandleWalkover(Tournament tournament)
{
    // Getting all matches in the first round
    List<Match> matches = await GetFirstRoundMatches(tournament);

    // Looping through all matches in the first round
    foreach (var match in matches)
    {
        if (match.Team1.Team == null)
        {
            int spotInNextMatch = 2;
            if (match.RoundIndex % 2 == 0)
                spotInNextMatch = 1;

            await MoveWinnerToNextMatch(tournament.Id, match.Team2, match.RoundIndex / 2, match.Round + 1, spotInNextMatch);
        }
        else if (match.Team2.Team == null)
        {
            int spotInNextMatch = 2;
            if (match.RoundIndex % 2 == 0)
                spotInNextMatch = 1;

            await MoveWinnerToNextMatch(tournament.Id, match.Team1, match.RoundIndex / 2, match.Round + 1, spotInNextMatch);
        }
    }
    _db.SaveChanges();
}

```

Figur 6-107 Metoden som håndterer kamper i første runde som vinner på walkover

#### 6.4.4.4.5 Kodelinje 5

På kodelinje fem blir turneringsstatusen satt lik 1.

#### 6.4.4.5 Starte turnering

```

[HttpGet("StartTournament")]
public async Task<ActionResult> StartTournament(string tournamentId)
{
    ...
}

```

Figur 6-108 Identifikasjon for StartTournament-endepunkt

Når en turnerings-admin ønsker å starte en turnering blir det gjort en get-forespørsel til endepunktet «/Tournament/StartTournament» med turneringsId som parameter. Da vil turneringsstatus endres til 2. Dette er bare mulig hvis turneringsstatus allerede er lik 1.

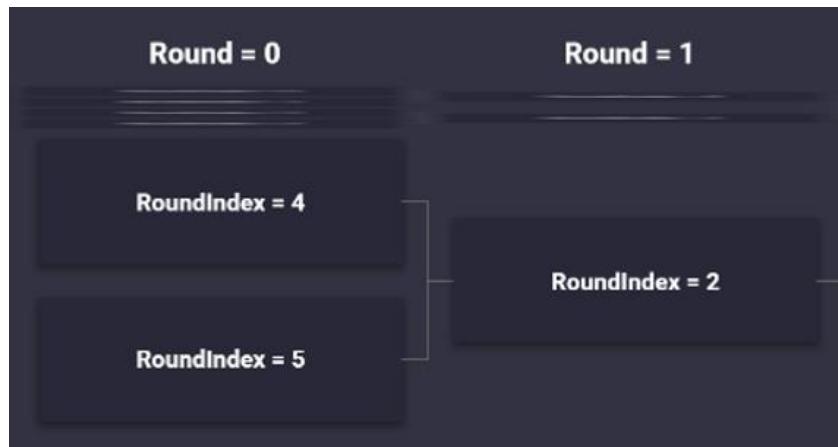
#### 6.4.4.6 Melde resultat

```
[HttpPost("SubmitResult")]
public async Task<ActionResult> SubmitResult(SubmittedResultModel submittedResult)
```

Figur 6-109 Identifikasjon for SubmitResult-endepunkt

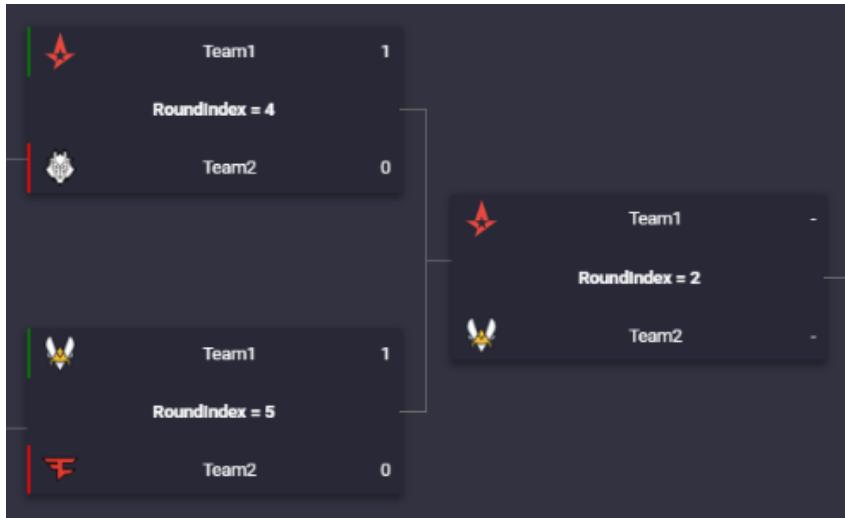
Når det blir deklarerer en vinner av en kamp blir det vinnende laget flyttet til Match-objektet med  $Round + 1$  og  $RowIndex / 2$ . Her benyttes heltallsdivisjon hvor svaret alltid blir rundet ned til nærmeste heltall, Det vil si at  $\frac{4}{2} = 2$  og  $\frac{5}{2} = 2$ , siden  $5 = 2 \cdot 2 + 1$ , hvor 1 er rest.

Vinneren av kampen hvor  $Round = 0$  og  $RowIndex = 4$ , blir flyttet til kampen med  $Round = 1$  og  $RowIndex = 2$ . Tilsvarende blir vinneren av kampen hvor  $Round = 0$  og  $RowIndex = 5$  blir flyttet til kamp med  $Round = 1$  og  $RowIndex = 2$ . Dette vises i Figur 6-110.



Figur 6-110 Rundeindeks fra runde 1 til runde 2

Match-Objektene blir opprettet med plass til to lag: Team1 og Team2. Vinneren av forrige kamp blir definert som Team1 eller Team2 basert på om heltallsdivisjonen ender med rest eller ikke. Det vil si at vinneren av kampen, fra eksempelet over, med  $RowIndex = 4$  blir Team1 og vinneren av kampen med  $RowIndex = 5$  blir Team2 i neste kamp. Dette vises i Figur 6-111.



Figur 6-111 Forrige bilde med lag

## 6.4.5 SignalR - Oppdatering mellom klienter i sanntid

To av kravene turneringsplattformen måtte innfri var en chat, og varsler. For at disse skal fungere måtte applikasjonen kunne oppdatere innholdet på flere klienter i sanntid. For å løse dette problemet har vi valgt å bruke ASP.NET biblioteket SignalR. Først kommer vi til å redegjøre for hvordan SignalR fungerer, så kommer vi til å gå i mer detalj rundt hvordan vi har valgt å implementere SignalR på turneringsplattformen.

### 6.4.5.1 Hubs

For at klienten og serveren skal prate sammen gjennom SignalR må man sette opp noe som kalles en «hub». Hubben lever på serveren, og inneholder metoder som blir brukt for å sende meldinger til klientene på applikasjonen i sanntid. Hubben har forskjellige objekter som gjør dette mulig. Eksempler på disse objektene er «Clients» og «Groups». For at en klient skal kunne kalle metoder fra hubben, må de være koblet opp mot den. Hubben kan bare sende meldinger til tilkoblede klienter. Mer om dette i neste avsnitt.

#### 6.4.5.2 Connection

Når en klient kobler seg til applikasjonen, åpnes alltid en tilkobling til hubben. Dette vises i Figur 6-112. Som nevnt i forrige avsnitt betyr dette at alle klienter kan kalle metoder fra hubben, og hubben kan sende meldinger til alle klienter i sanntid. Figur 6-113 viser metoden «SendMessage», som ligger i hubben. Denne bruker Clients-objektet for å aksessere alle tilkoblinger gjennom «Clients.All». «SendAsync» metoden til Clients.All sender en melding med variabler til alle klienter koblet opp til hubben. SendMessage metoden i Figur 6-113 sender altså meldingen «ReceiveMessage» med variabelen «message» til alle tilkoblede klienter i sanntid.

```
const connection = new HubConnectionBuilder()
    .withUrl(`/hubs/chat`)
    .withAutomaticReconnect()
    .build();

await connection.start();
this.setState({ connection });
```

Figur 6-112: Klient kode: En klient starter en åpen tilkobling til serveren gjennom en hub

```
public Task SendMessage(string message)
{
    return Clients.All.SendAsync("RecieveMessage", message);
}
```

Figur 6-113 Metode "SendMessage" i en hub

For at klienten skal reagere på meldinger sendt fra hubben kan man be klienten lytte etter spesifikke meldinger over tilkoblingen sin. Dette vises i Figur 6-114. Når klienten plukker opp meldingen «RecieveMessage» oppdaterer klienten innholdet sitt. Siden metoden i Figur 6-113 Metode "SendMessage" i en hub sendte ut «RecieveMessage»

til alle tilkoblede klienter vil alle klienter oppdatere innholdet sitt i sanntid. Message-variabelen som blir sendt fra hubben i Figur 6-113 er den samme message-variabelen som i Figur 6-114.

```
useEffect(() => {
  if (connection) {
    try {
      connection.on("ReceiveMessage", (message) => {
        const updatedChat = [...latestChat.current];
        updatedChat.push(message);

        setChat(updatedChat);
      });
    } catch (e) {
      console.log("Connection failed: ", e);
    }
  }
}, [connection]);
```

Figur 6-114 Klienten som lytter etter en melding sendt over tilkoblingen

#### 6.4.5.3 Hubs implementert i kontrolleren

Hver gang klienten får tilsendt en melding over tilkoblingen sin vil man ofte gjøre mer enn å bare forandre på sidens innhold. Når det blir sendt en melding i en chat, eller man får et varsel ønsker man også å lagre disse i databasen, slik at det oppdaterte innholdet på siden ikke forsvinner ved en refresh. Siden applikasjonen vår bruker lagdeling, skal ikke hubben prate med DALen. En løsning er for en klient å kalle en controller-metode, for så å kalle en hub-metode. Dette er ikke så ryddig da klienten må gjøre to serverkall. Derfor valgt å implementere et interface av typen `IHubContext<HubName>` inn i flere kontrollere. Dette gjør så vi kan bruke alle fordelene til en hub, som å sende meldinger gjennom Client-objektet, i en kontroller. Dermed kan man sende meldinger til tilkoblinger, fra en kontroller, samtidig som man utfører databasetransaksjoner gjennom DALen. Dermed kan man oppdatere databasens innhold, og sende meldinger til klienter i samme metode. For å bruke

Clients-objektet i en kontroller kan man ikke kalle dette direkte som i en hub, men ved å bruke «\_hubContext»-objektet. Å sende en melding til alle tilkoblede klienter i en kontroller gjøres da som dette: \_hubContext.Clients.All.SendAsync(). Figur 6-115 viser hvordan dette interfacet blir implementert.

```
[Route("[controller]")]
[ApiController]
public class SignalRController : ControllerBase
{
    private IHubContext<ChatHub> _hubContext;
    private readonly ISignalRRepository _signalr;
    private readonly IUserRepository _user;
    private readonly ITournamentRepository _tournament;
    private readonly ITeamRepository _team;
    private const string _userIdKey = "UserId";

    public SignalRController(IHubContext<ChatHub> hubContext, ISignalRRepository signalrDB, IUserRepository userDB)
    {
        _hubContext = hubContext;
        _signalr = signalrDB;
        _user = userDB;
        _tournament = tournamentDB;
        _team = teamDB;
    }
}
```

Figur 6-115 Implementering av hub-interface i en controller

#### 6.4.5.4 Grupper for å sende meldinger

Hittil har det blitt vist hvordan man sender ut en felles melding til alle tilkoblede klienter. Vi ønsker å kunne sende ut meldinger til spesifikke klienter. Dette kan for eksempel være en gruppe med klienter, som alle har logget inn på samme bruker. Dette har vi valgt å løse gjennom grupper i SignalR. En SignalR-gruppe inneholder flere tilkoblinger. Man kan da sende en melding til alle klientene med tilkoblinger i SignalR-gruppen. Dette gjøres med SendAsync metoden som tidligere, bare denne gangen gjennom «\_hubContext.Clients.Groups(groupname)».

Her vises et eksempel på hvordan vi har brukt SignalR-grupper for å delegere varsler. Mer spesifikt blir det gjennomgått hvordan alle klientene logget inn med en bruker, kan få private varsler i sanntid

Figur 6-116 viser hvordan Users-objektet som former Users-tabellen i databasen, ser ut. Som den røde boksen viser så inneholder Users-objektet en liste med «Notificationgroups». En notifikasjonsgruppe er et objekt som inneholder en id, og et

navn. Figur 6-118 er et utklipp av prosessen hvor en bruker registrerer seg. Ved registrering får brukeren lagt til en notifikasjonsgruppe, hvor brukerens id blir brukt som gruppens navn. Idet blir alltid brukt som Notifikasjonsgrupper og SignalR-gruppers navn, da de er unike, og det er forutsigbart hva en gruppe kommer til å hete. For en kamprom-gruppe ville kamprom-Iden blitt brukt som gruppenavn.

```
public class Users
{
    public Guid Id { get; set; }
    public DateTime RegisterDate { get; set; }
    public string Email { get; set; }
    public string Username { get; set; }
    public byte[] Password { get; set; }
    public byte[] Salt { get; set; }
    public string Bannerfilename { get; set; }
    public string ProfileImagefilename { get; set; }

    public string ActivationCode { get; set; }
    public DateTime ActivationCodeExpire { get; set; }

    public int LoginAttempts { get; set; }
    public DateTime DeactivatedUntil { get; set; }

    public string Cookie { get; set; }

    public string SteamId { get; set; }

    public string Discord { get; set; }
    virtual public IList<NotificationGroups> NotificationGroups { get; set; }
}
```

```
public class NotificationGroups
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Figur 6-116 Alle brukere har en liste med notifikasjonsgrupper i db

Figur 6-117 Notifikasjonsgruppe inneholder Id og navn

```
NotificationGroups personalGroup = new NotificationGroups { Name = userId.ToString() };

Users newUser = new Users
{
    Id = userId,
    Email = userModel.Email,
    Username = userModel.Username,
    Password = passwordHash,
    Salt = salt,
    Bannerfilename = "profileBannerPlaceholder.png",
    ProfileImagefilename = "profileImagePlaceholder.png",
    ActivationCode = null,
    RegisterDate = DateTime.Now,
    NotificationGroups = new List<NotificationGroups>{ personalGroup }
};

_generic.Create(newUser);
await _generic.SaveChangesAsync();
return Ok(userId);
```

Figur 6-118 Ved opprettning av bruker blir det lagt til en notifikasjonsgruppe med navn likt som brukerens ID

Etter at en bruker er blitt registrert blir den automatisk logget inn. Når en bruker blir logget inn blir metoden i Figur 6-119 kjørt. Da blir listen med notifikasjonsgrupper tilhørende brukeren gått gjennom, og klienten blir lagt til SignalR-grupper med tilsvarende navn som notifikasjonsgruppene denne brukeren tilhører. Dette inkluderer brukerens personlige notifikasjonsgruppe, og alle andre notifikasjonsgrupper brukeren er med i. Det vil si at om du er innlogget med samme bruker på forskjellige klienter vil alle klientene motta meldinger fra serveren i sanntid.

```
[HttpGet("JoinAllNotificationGroups")]
public async Task<ActionResult> JoinAllNotificationGroups(string connectionId)
{
    try
    {
        string userId = HttpContext.Session.GetString(_userIdKey);
        if (userId == null)
            return BadRequest("Not signed in");

        Users user = await _user.GetUserById(userId);

        int i = 0;
        while (i < user.NotificationGroups.Count)
        {
            await _hubContext.Groups.AddToGroupAsync(connectionId, user.NotificationGroups[i].Name);
            i++;
        }
        return Ok(user);
    }
    catch
    {
        return BadRequest("Could not join group");
    }
}
```

Figur 6-119 Alle brukerens notifikasjonsgrupper blir gått gjennom, og klienten blir lagt til SignalR-gruppe med samme navn som notifikasjonsgruppen

For å sende et privat varsel til den nylig opprettede brukeren, brukes SendAsync-metoden til `_hubContext.Clients.Group`. Et eksempel på dette vises i Figur 6-120. Klienter logget inn som denne brukeren får en melding «Notification» sammen med et notification-objekt, som vises i Figur 6-121. Klienten(e) oppdaterer deretter innholdet i sanntid, og viser varselet.

```
private async Task SendGroupNotification(GroupNotifications notification)
{
    await _hubContext.Clients.Group(notification.Groupname).SendAsync("Notification", notification);
    await _signalr.SaveNotification(notification);
}
```

Figur 6-120 Metode på serversiden som sender notifikasjon til gruppe

```
connection.on("Notification", (notification) => {
    updateNotifications(notification, "add");
});
```

Figur 6-121 Klienten som lytter etter meldingen "Notification", og oppdater siden om dette blir plukket opp

## 6.4.6 Sikkerhet

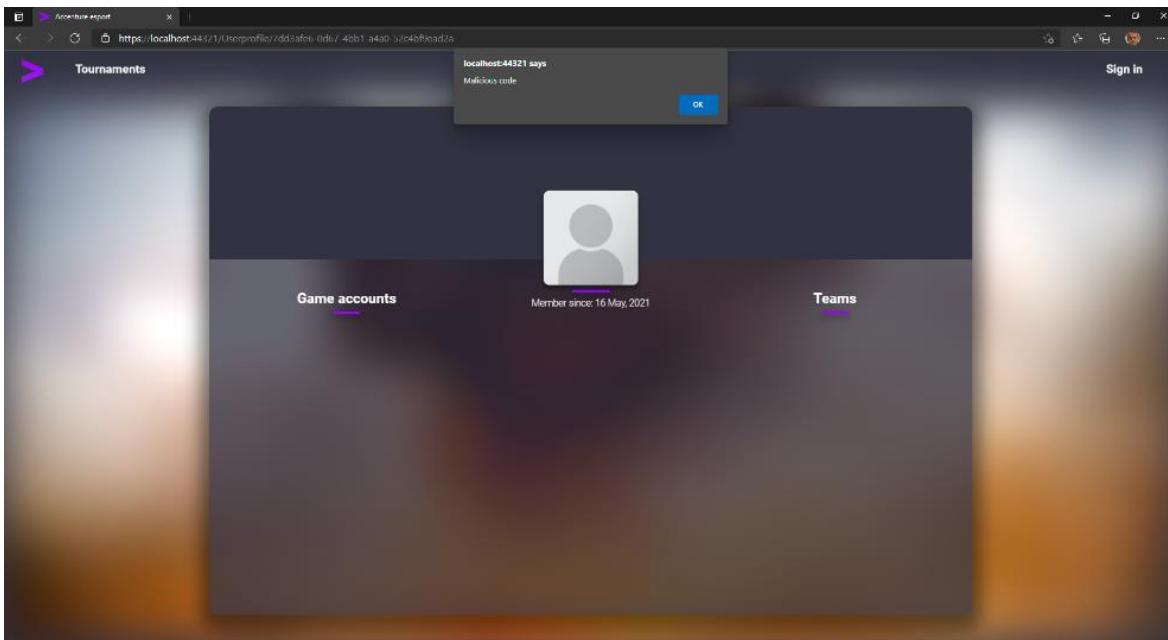
For å skille mellom ulike brukere så inneholder applikasjonen vår brukerregistrering. For å hindre at ondsinnede brukere får tilgang til sensitiv data, benytte funksjonalitet de ikke har rettighet til eller utgir seg for å være noen andre så har vi implementert en rekke sikkerhetstiltak. I denne delen skal vi redegjøre for ulike trusler en webapplikasjon står ovenfor, og hvilke tiltak vi har implementert for å håndtere disse truslene.

### 6.4.6.1 Cross Site Scripting (XSS)

XSS er et begrep for en teknikk hvor en ondsinnet bruker sender inn kode, forkledd som vanlig input data, som blir lagret i databasen. Når en annen bruker henter ut denne dataen så tolker ikke nettleseren det som tekst, men som kode og utfører det den ondsinnede brukeren skrev i koden.

Et eksempel på dette er hvis en bruker registrerer seg med brukernavnet: alert(«Malicious code»). Alert er en Javascript-funksjon som åpner en varseldmelding med en gitt tekst i nettleseren til brukeren. Alle som besøker denne brukerens profil eller på noen som helst annen måte blir vist denne brukeren sitt brukernavn vil få opp denne irriterende varseldmeldingen. Dette eksempelet er harmløst, men det kan virkelig

skape problemer hvis scriptet for eksempel er skrevet for å slette en turnering eller melde inn feil kampresultat.

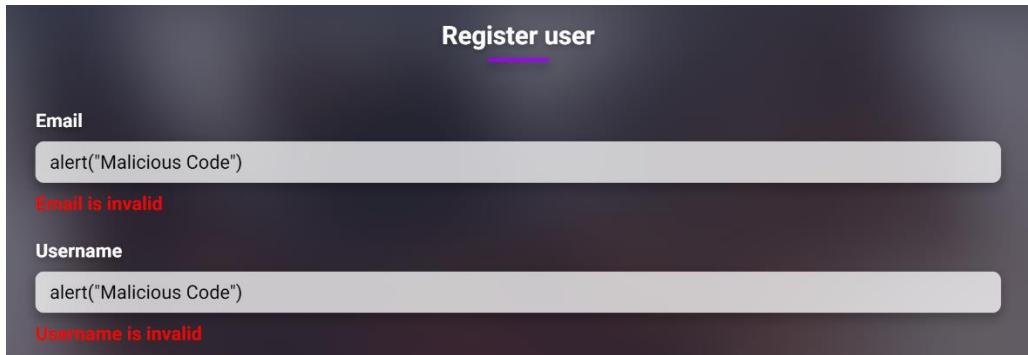


Figur 6-122 Brukerprofil med varselsboks

#### 6.4.6.1.1 Demo av Cross Site Scripting

Vi har satt inn en rekke tiltak for å hindre XSS. Først og fremst har vi satt restriksjoner i klientkoden for hva slags tegn brukeren skal ha mulighet til å inkludere. Dette er mest for å bedre brukeropplevelsen etter som det er mulig å benytte diverse teknikker for å omgå valideringen i klientkoden. Videre har vi tilsvarende inputvalidering i serverkoden som brukeren ikke har mulighet til å omgå. Vi benyttet oss av «Regular Expression» (regex) for å gjennomføre valideringen. Kort forklart så er regex forhåndsprogrammerte

mønstre hvor man definerer hva en tekst kan og ikke kan inneholde. Vi benyttet identisk regex for tilhørende datafelter på klientsiden og server<sup>3</sup>siden.



Figur 6-123 Inputvalidering i brukergrensesnittet

```
public class UserModel
{
    [RegularExpression(@"^([a-zA-Z0-9_\-.]+@[a-zA-Z0-9_\-.]+\.( [a-zA-Z]{2,5})$")]
    public string Email { get; set; }

    [RegularExpression(@"^([a-zA-Z0-9_ -]{2,20}$")]
    public string Username { get; set; }

    [RegularExpression(@"^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9]).{8,30}$")]
    public string Password { get; set; }
}
```

Figur 6-124 Inputvalidering på server

#### 6.4.6.2 SQL-Injections

SQL-Injections har noen fellestrek med XSS ved at en ondsinnet bruker sender inn kode forkledd som input data. Forskjellen er at der XSS sitt formål er å eksekvere koden i en annen bruker sin nettleser, så har SQL-Injections som mål å skade databasen direkte. Transaksjoner med SQL baserte database foregår ved hjelp av spørringsspråket SQL. Hvis den ondsinnede brukeren klarer å sende inn data som gjør modifikasjoner til disse spørringene kan den skape store problemer for databasen sin

---

<sup>3</sup> Se ordliste kapittel 9

integritet, tilgjengelighet og/eller konfidensialitet. Dette ved å for eksempel modifisere, slette eller lese data den ikke skal ha tilgang til. Det er heldigvis flere måter å beskytte seg mot slike angrep. Vi valgte å benytte et abstraksjonslag gjennom Entity Framework Core, slik at vi aldri snakker direkte med databasen gjennom SQL spørninger. Dette skrives om i kapittel 6.4.2.3.

#### **6.4.6.3 Brute-force angrep**

Det er et problem at brukere velger for svake passord som er veldig sårbare for brute-force angrep, hvis det ikke er restriksjoner på plass. Når en bruker registrerer seg på plattformen blir den tvunget til å velge et passord med minst åtte tegn, en stor bokstav og et tall. Et brute-force-angrep i denne sammenhengen betyr at en ondsinnet aktør benytter en maskin med høy prosessorkraft for å systematisk teste ulike brukernavn og passord kombinasjoner helt til riktig kombinasjon blir funnet. For å sikre oss mot dette har vi implementert et mottiltak ved at en brukerkonto blir deaktivert i 15 minutter etter fem feilaktige innloggingsforsøk på rad.

#### **6.4.6.4 Hashing og salting av passord**

Når en ny bruker registrerer seg på plattformen, blir brukerinformasjonen sendt fra klienten til serveren. Deretter blir det opprettet et salt, som er en tilfeldig generert sekvens med bytes. Videre blir både saltet og passordet til brukeren benyttet som input i en krypteringsalgoritme som er matematisk bevist å være umulig å reversere med dagens teknologi. Den krypterte passord-salt kombinasjonen blir lagret i databasen sammen med det originale saltet.

Når brukeren ønsker å logge seg inn blir først passordet og brukernavnet/e-posten sendt til serveren. Deretter blir saltet og den krypterte passordet-salt kombinasjonen, som er knyttet til brukeren som matcher brukernavnet/e-posten, hentet fra databasen. Videre blir det innsendte passordet og det eksisterende saltet benyttet som input, i den samme krypteringsalgoritmen som ble nevnt i avsnittet over, for å generere en ny passord-salt kombinasjon. Til slutt blir den eksisterende passord-salt kombinasjonen

fra databasen sammenlignet med den nye, og hvis de er like betyr det at brukeren har oppgitt korrekt passord. På denne måten er det aldri nødvendig å oppbevare brukerne sine passord i klartekst.

```
public static byte[] CreateSalt()
{
    var csp = new RNGCryptoServiceProvider();
    var salt = new byte[24];
    csp.GetBytes(salt);
    return salt;
}
```

Figur 6-125 Algoritmen for å generere et salt

```
public static byte[] CreateHash(string password, byte[] salt)
{
    return KeyDerivation.Pbkdf2(
        password: password,
        salt: salt,
        prf: KeyDerivationPrf.HMACSHA512,
        iterationCount: 1000,
        numBytesRequested: 32);
}
```

Figur 6-126 Krypteringsalgoritmen

#### 6.4.6.5 Autentisering på server

Ved en vellykket innlogging blir brukeren sin unike ID lagret som en session-variabel på serveren. Denne blir av sikkerhetshensyn slettet etter 1 time uten aktivitet. Brukeren kan imidlertid veldig å forbli innlogget, da blir en cookie lagret i brukeren sin nettleser, og automatisk benyttet istedenfor å manuelt skrive brukeridentitet og passord hver gang brukeren besøker plattformen.

Hver gang klienten til brukeren kommuniserer med et endepunkt på serveren hvor det krever innlogging for å få tilgang, blir session-variabelen som er knyttet til den

spesifikke klienten aksessert. Koden i Figur 6-127 er inkludert i starten av alle endepunkter hvor dette er tilfelle. I første kodelinje blir IDen til brukeren aksessert fra session-variabelen. I linje to blir det testet om denne er definert eller ikke. Er den ikke definert blir tredje kodelinje utført, denne sender en tilbakemelding til brukeren at den ikke er innlogget. Hvis IDen er definert betyr det at brukeren er innlogget, og resten av koden blir utført.

```
string userId = HttpContext.Session.GetString(_userIdKey);
if (userId == null)
    return Unauthorized("Not signed in");
```

Figur 6-127 Sjekk for om brukeren er innlogget

Webapplikasjonen inkluderer funksjonalitet som, i tillegg til innlogging, krever at brukeren er autorisert. Et eksempel er at det kun er brukere som er turneringsadmin for en spesifikk turnering som skal ha rettighet til å gjøre endringer på denne turnering. De to figurene Figur 6-128 og Figur 6-129 under viser koden som er inkludert for å autentisere en bruker som administrator for en spesifikk turnering.

```
string userId = HttpContext.Session.GetString(_userIdKey);
if (userId == null)
    return Unauthorized("Not signed in");

List<Users> touramentAdmins = await _tournament.GetTournamentAdmins(tournamentId);
if (!Auth.HasAccess(userId, touramentAdmins))
    return Unauthorized("User is not tournament admin");
```

Figur 6-128 Kode for å autentisere en bruker som turneringsadministrator

```
public static bool HasAccess(string userId, List<Users> users)
{
    bool hasAccess = false;
    foreach (var user in users)
    {
        if (userId == user.Id.ToString())
        {
            hasAccess = true;
            break;
        }
    }

    return hasAccess;
}
```

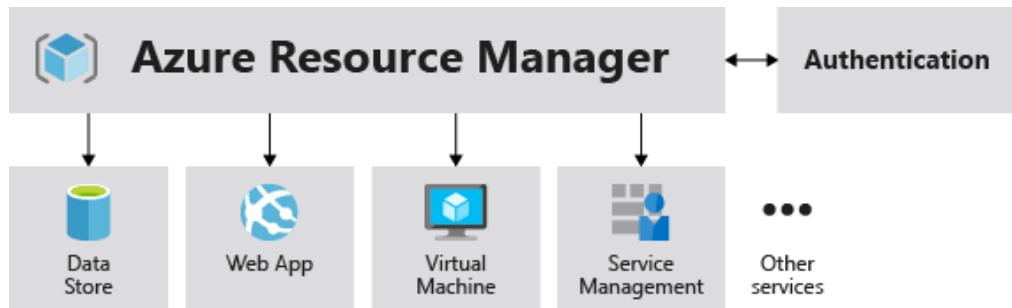
Figur 6-129 Kode for å sjekke om en av brukerne i en liste har en gitt ID

## 6.5 Deployment og hosting

For deployment og hosting bruker vi Microsoft Azure (Azure), siden IDEen vi brukte for å utvikle serversiden vår (Microsoft Visual Studio) har integrert funksjonalitet for å deploye til denne tjenesten.

### 6.5.1 Microsoft Azure

For å kunne bruke Microsoft Azure for deployment måtte gruppen først opprette en «Resource group». Microsoft Azure har mange forskjellige ressurser, og en resource group er en kontainer hvor du kan samle alle Azure ressursene som blir brukt til et prosjekt.



Figur 6-130 Azure resource group illustrasjon (Tfitzmac et al.)

Name	Type	Location	Actions
accentureesport	Azure Cosmos DB account	Norway East	...
accentureesport	SQL server	Norway East	...
accentureesportportal	App Service	West Europe	...
BrukerTesting	App Service	West Europe	...
DanielTesting	App Service	West Europe	...
esportDB (accentureesport/esportDB)	SQL database	Norway East	...
esportstoragecontainer	Storage account	Norway East	...
hermanTesting	App Service	West Europe	...

Figur 6-131 Utklipp av vår Azure Resource Group

## 6.5.2 Azure App Service – Web App

Etter å ha opprettet en resource group i Microsoft Azure måtte vi lage en Azure App Service av typen Web App. Dette er en HTTP-basert tjeneste som lar deg publisere web applikasjoner. Azure App Service kommer med forskjellige fordeler som load balancing, autoscaling, og dens DevOps funksjonalitet som Continuous Delivery fra f.eks. Github.

Da vi jobbet i forskjellige Git-Hub brancher, måtte vi til tider publisere forskjellige branches for testing til samme tid. Dermed lagde vi totalt fire Web Apps. En for vår Main Branch i Github, to for testing og en for brukertestning.

## 6.5.3 Azure pipelines continuous delivery

Til webappen som tilhørte Main branchen implementerte vi continuous delivery gjennom Azure Pipelines. Dette var koblet opp mot Github-repositoriet vårt, slik at hver gang vi pushet til Main branchen vår, så ble dette automatisk publisert. Linken til denne Web Appen ga vi til veiledere og produkteier, slik at de hadde muligheten til å følge progresjonen vår om de ønsket det.

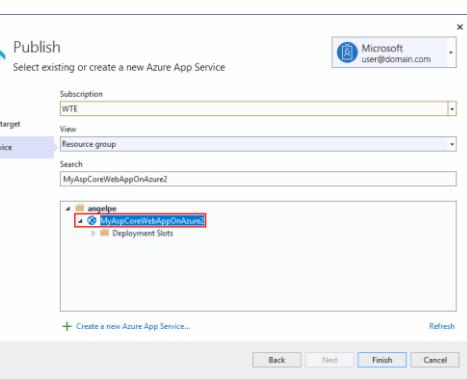
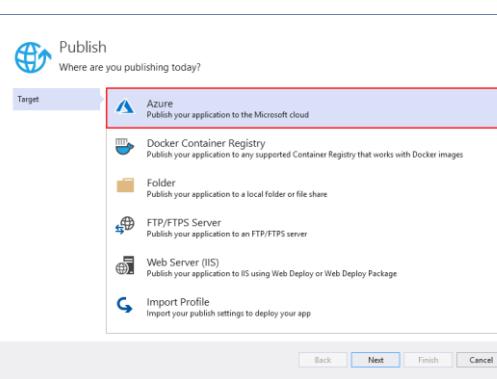
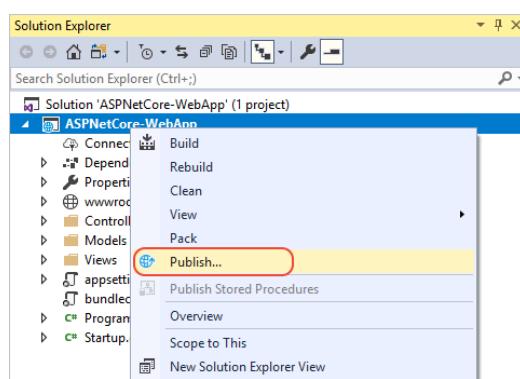
The screenshot shows the Azure DevOps Pipelines interface for the 'accentureesportportal' project. The left sidebar has 'Pipelines' selected. The main area shows a list of recent pipeline runs under the 'Runs' tab. Each run is listed with its description, stage status, and execution details (date, duration). The runs are:

Description	Stages	Date	Duration
#20210507.2 Delete sysimg.jpg	1 green checkmark	May 7	5m 1s
#20210507.1 Add files via upload	1 green checkmark	May 7	6m 36s
#20210504.2 Se bort forrige push	1 green checkmark	May 4	5m 20s
#20210504.1 Tester opplasting av bilder	1 green checkmark	May 4	5m 54s
#20210418.4 Update azure-pipelines.yml for Azure Pipelines	1 green checkmark	Apr 18	-

Figur 6-132 Azure pipelines som viser kjøringer når det er pushet kode til main-branchen

## 6.5.4 Visual Studio Publish

For Web Appene som ikke brukte Azure Pipeline og continuous delivery måtte vi manuelt publisere oppdateringene våre hver gang. Dette ble gjort gjennom en funksjonalitet i Visual Studio som heter «Publish». Ved å logge inn med samme e-post som blir brukt på Azure får man muligheten til å velge hvilken App Service man ønsker å laste opp til. Dette er illustrert nedenfor.



Figur 6-133 Publisere til Azure app service steg 1

Figur 6-135 Publisere til Azure app service steg 2

Figur 6-134 Publisere til Azure app service steg 3

## 6.5.5 Database på Azure Server

Da vi publiserte prosjektet til Azure app service benyttet den en skrivebeskyttet in-memory database. Det var ikke optimalt siden vi ønsket å kunne gjøre endringer direkte for testing, slik vi kunne via DB browser lokalt. Siden vi på publiseringstidspunktet lagret bildene i en mappe på klientdelen (ClientApp), kunne vi ikke laste opp bildene siden alle mapper ble skrivebeskyttet etter publisering på Azure. I slutten av april fikk gruppen et møte med en Accenture-ansatt med god kompetanse innen publisering og vedlikehold på Azure. Han foreslo at vi skulle se på tjenesten Cosmos DB. Det er en No SQL-database, som kan håndtere flere typer innhold. Med denne løsningen kunne vi håndtere bildene og database sammen.

Han nevnte også at et alternativ var å bruke Azure SQL database og blob storage for bilder via en storage account.

Gruppen ønsket å få Cosmos DB til å fungere, men etter å lest oss opp måtte vi innse at vi ikke hadde tiden til å skrive om koden i vår i DALen. Da rettet vi fokuset angående å deploye og hoste webappen mot en Azure SQL database og en storage account for bilder.

# 7 Testdokumentasjon

## 7.1 Innledning

Vi hadde en plan om å skrive automatiserte enhetstester. I prosessdokumentasjonen har vi allerede skrevet at enhetstestene måtte skrives om i sprint 3 på grunn av refakturert kode. Da bestemte vi også for å gå over til manuell enhetstesting for å holde tidsskjemaet.

Gruppen er ikke kjent med automatiserte integrasjonstester i .NET Core, så her brukte vi også manuell testing. Vi har ført opp feil som må utbedres fortløpende i vårt arbeidsfordelingsverktøy Jira. Vi har også opprettet flere Azure App services for at hvert gruppemedlem skal kunne teste webapplikasjonen i et produksjonsmiljø.

## 7.2 Brukertestene

Brukertestene ble gjennomført over Microsoft Teams 12. mai. Det var fem brukertester med ansatte fra Accenture Norge. Vi hadde planlagt en sjette brukertest, men vedkommende fikk en flyavgang utsatt og måtte dermed avlyse brukertesten. Han skrev at han kunne gjennomføre brukertesten på et senere tidspunkt. På grunn av tidspress valgte vi å droppe dette. Det var to dager før vi skulle avslutte utviklingen. Dermed visste vi at de aller fleste tilbakemeldinger først og fremst er nyttige for teamet som skal videreutvikle denne plattformen i fremtiden.

Brukertesterne skulle gjennom følgende program:

1. Registrer bruker
2. Endre brukernavn
3. Bytte profilbilde
4. Opprett lag

5. Endre lagnavn
6. Inviter brukeren «dev1ce» til laget
7. Melde laget sitt på en turnering
8. Søk etter brukeren: «sherman»
  
9. Godta invitasjon til et lag, som vi sender ut
  
10. Logge inn med brukernavn: dev1ce og passord: Tester123, som spiller for NIP
11. Finne fram til neste kamp NIP skal spille i turneringen: AccentureLan 2021
12. Meld at laget klar til kamp
13. Meld inn seier med resultatet 1 - 0 til NIP
14. Opprette en turnering

Vi kan kategorisere oppgavene i følgende kategorier:

- 1: registering
- 2-3: brukerprofil
- 4-6: lagprofil
- 7: turneringspåmelding
- 8: søkefunksjonalitet
- 9: notifikasjon
- 10-13: kampflyt
- 14: turneringsopprettelse

Første brukertest kjørtes via et nettsted hostet med en Azure app service. Resten av brukertestene måtte bli gjort ved kjøring lokalt hos en av gruppemedlemmene, mens brukertesteren tok kontroll over dette gruppemedlemmets musepeker. Grunnen til at vi ikke kunne bruke nettstedet hostet med en Azure app service var at vår gratis kredit i Azure ble brukt opp og app servicen ble følgelig stoppet. App servicen ble stoppet mellom brukertest en og brukertest to. Da måtte vi være snartekte og fant ut at en møtedeltager på Teams kan ta over musepekeren. Dermed kunne vi gjennomføre de fire siste brukertestene lokalt hos en av gruppemedlemmene.

## 7.2.1 Gjennomføring

### 7.2.1.1 Brukertest 1

**Erfaringsnivå:** Testpersonen er en trent bruker. Han er medlem av spillgruppen og Han kjenner til prosjektet.

**Registrering:** Testpersonen klarer oppgaven fint og har ingen innspill

**Brukerprofil:** Testpersonen klarer oppgavene fint og har ingen innspill. Han bruker «pluss»-knapp for å opprette et lag.

**Lagprofil:** Testpersonen klarer oppgavene fint og har ingen innspill.

**Turneringspåmelding:** Testpersonen klarer oppgaven fint og har ingen innspill. Han trykker inn på turneringen istedenfor for å bruke «Join»-knappen fra turneringskortet.

**Søkefunksjonalitet:** Testpersonen klarer oppgaven fint og har ingen innspill.

**Notifikasjon:** Denne oppgaven ble ikke gjennomført. (Vi hadde ikke planer om å gjøre denne oppgaven på daværende tidspunkt på grunn av tekniske problemer. Det endret seg da vi kunne gjøre brukertesten lokalt.)

**Kampflyt:** Testpersonen finner turneringstreet, men trykker på en annen kamp enn den han er instruert til å finne. Han går raskt tilbake og finner neste kamp til laget «NIP». På grunn av tekniske problemer med databasen lot ikke alle oppgaver seg gjøre. Han viser forståelse for dette og brukertesten gikk til neste oppgave.

**Turneringsopprettelse:** Testpersonen klarer oppgaven fint. Han oppdager at Regex på turneringstittelen er for streng og skjemaet krever at det rettes før man kan gå videre. Vår Regex på feltet tillot ikke tegnet «:». Han skjønner siden med turneringsfaser i registreringsskjemaet, men synes det er uklart om man er turneringsadministrator for egen turnering. Det bør komme tydelig fram underveis i prosessen. Han sier at det ikke er helt tydelig at det kun er «Single elimination» som

er mulig turneringsformat. Dessuten burde ikke «Bracket»-knappen vises hvis ikke turneringstreet er laget.

**Generell tilbakemelding:** Han er veldig imponert og synes webappen er godt og intuitivt laget.

### 7.2.1.2 Brukertest 2

**Erfaringsnivå:** Testpersonen er en nybegynner og kjenner ikke til prosjektet.

**Registrering:** Testpersonen finner «Sign up»-knappen på «logg inn»-siden, men savner en egen «Sign up»-knapp. Hun skriver inn et for svakt passord, men systemet gir for dårlig tilbakemelding, så vi må fortelle hvilke regler som gjelder for et passord.

**Brukerverte:** Testpersonen klarer oppgavene fint og har ingen innspill. Hun bruker navbarknappen «Create team» for å opprette lag. Hun synes at laget skal komme opp med en gang på brukerprofilen etter opprettelse. Siden må lastes inn på nytt for å vise laget på profilsiden.

**Lagprofil:** Testpersonen klarer å finne laget. Når hun har søkt opp «dev1ce» for å legge han til på laget, prøver hun å trykke på plussknappen før profilbildet er lastet inn. Akkurat når hun trykket på plussknappen lastet profilbilde inn. Hun trykket dermed på profilbildet og ble tatt til profilen til «dev1ce». Hun går deretter tilbake og venter til profilbilde på søkermodalen er lastet inn. Dette problemet er bare tilstede når man kjører opp prosjektet lokalt. Hun mener at en «pop out»-modal hadde vært veldig fint å legge til når man skal se på en brukerprofil fra søkerinduet.

**Turneringspåmelding:** Testpersonen klarer oppgaven fint og har ingen innspill. Hun trykker inn på turneringen og bruker «Join»-knappen der istedenfor for å bruke «Join»-knappen fra turneringskortet.

**Søkefunksjonalitet:** Testpersonen klarer oppgaven fint og har ingen innspill.

**Notifikasjon:** Testpersonen klarer oppgaven fint og har ingen innspill.

**Kampflyt:** Testpersonen finner ikke turneringstreet, og trenger hjelp til dette. Når hun har turneringstreet foran seg finner hun ikke riktig kamp. Hun vil ha en visuell markering på neste kamp for laget. Dessuten ønsker hun dato for kampen.

Når hun kommer til oppgaven om å melde kampresultat oppdager hun en svakhet ved modalen for dette. Hun kan ikke skrive inn resultat rett inn, men må markere «placeholderen», før resultatet kan skrives inn.

**Turneringsopprettelse:** Testpersonen klarer oppgaven fint. Hun sliter med å skjønne siden med turneringsfaser i registreringsskjemaet.

**Generell tilbakemelding:** På bakgrunn av problemene med å skjønne hva neste kamp i turneringstreet er for et lag, foreslår vi at man kan få beskjed via en notifikasjon. Det er hun positiv til. Hun er veldig alt i alt positiv til webappen og synes den er fin.

### 7.2.1.3 Brukertest 3

**Erfaringsnivå:** Testpersonen er en nybegynner og kjenner ikke til prosjektet.

**Registrering:** Testpersonen finner ikke «Sign up»-knappen og må ha hjelp. Hun vil ha egen «Sign up»-knappen fordi hun ikke har en konto allerede. Hun skriver inn et for svakt passord, men systemet gir for dårlig tilbakemelding, så vi må fortelle hvilke regler som gjelder for et passord.

**Brukertilgang:** Testpersonen klarer å laste opp et bilde og endre brukernavn, men blir usikker når laget skal opprettes, men finner navbarknappen «Create team» for å opprette lag. Hun synes at laget skal komme opp med en gang på brukerprofilen etter opprettelse.

**Lagprofil:** Testpersonen blir veldig forvirret av lagprofilen og brukerprofilen fordi de ser så like ut. Vi spør om det burde være forskjellige «Placeholder»-bilder på de to sidene. Dette mener hun er en god idé. Når hun har søkt opp «dev1ce» for å legge han til på laget, trykker hun på plussknappen før profilbildet er lastet inn. Da kommer

hun til profilen til «dev1ce», men går tilbake og venter til profilbilde på søkermodalen er lastet inn.

**Turneringspåmelding:** Testpersonen sliter med å finne turneringsoversikten. Hun ønsker at «Create Team»-knappen i navbaren skal være ved siden av turneringsknappene, ikke mellom dem. Rekkefølgen vi har på knappene i navbaren er ikke spesielt logisk ifølge henne. Hun trykker inn på turneringen istedenfor for å bruke «Join»-knappen fra turneringskortet.

**Søkefunksjonalitet:** Testeren klarer oppgaven fint og har ingen innspill.

**Notifikasjon:** Testeren sliter litt med å finne notifikasjonsbjellen, men når hun finner den forstår hun notifikasjonens innhold og aksepterer laginvitasjonen. Hun liker notifikasjonen og har ingen ytterligere innspill.

**Kampflyt:** Testeren finner ikke turneringstreet, og trenger hjelp til dette. Resten av oppgavene får fint. Vi endret turneringstreet til å vise bare førsterundekamper. Det var ikke så logisk å finne en semifinalekamp. Akkurat som brukertester 2, påpeker hun svakheten med «melde resultat»-modalen.

**Turneringsopprettelse:** Testeren klarer oppgaven fint. Hun sliter med å skjønne siden med turneringsfaser i registreringsskjemaet. Hun påpeker at turneringsfasene kan stå på venstreside av antall kamper i disse fasene. Hun vil ha en bedre forklaring på hva boksene betyr i en «Tooltip». Hun trykker på en turneringsadministrator, som sender henne ut av registreringsskjemaet og hun mister alt hun har fylt inn. Ellers går registreringen fint.

**Generell tilbakemelding:** Hun er veldig alt i alt positiv til webappen og synes den er fin.

#### 7.2.1.4 Brukertest 4

**Erfaringsnivå:** Testpersonen er en nybegynner og kjenner ikke til prosjektet.

**Registrering:** Testpersonen finner ikke «Sign up»-knappen og må ha hjelp. Hun vil ha egen «Sign up»-knappen fordi hun ikke har en konto allerede. Hun påpeker at «Sign up»-knappen bør være på toppen av «Sign in»-siden for bedre synlighet med en tekst som for eksempel «Not a user yet? sign up here».

**Brukerprofil:** Testpersonen klarer oppgavene fint, men vil at modalen for å endre brukerinnstillingene skal lukke seg når man lagrer endringene. I tillegg er det ikke så lett å få øye bekreftelsesboksen vi viser når databasekall er vellykket. Denne boksen forsvinner litt for fort. Vi har satt den til og vises i tre sekunder. Hun synes at laget skal komme opp med en gang på brukerprofilen etter opprettelse.

**Lagprofil:** Testpersonen klarer oppgavene fint og har ingen innspill.

**Turneringspåmelding:** Testpersonen sliter med å finne turneringsoversikten. Hun ønsker at «Create Team»-knappen i navbaren skal være ved siden av turneringsknappene, ikke mellom dem. Hun sa at vi gjerne kunne bruke nedtrekksmeny på turneringsvalg i navbaren. Rekkefølgen vi har på knappene i navbaren er ikke spesielt logisk ifølge henne. Hun trykker inn på turneringen istedenfor for å bruke «Join»-knappen fra turneringskortet.

**Søkefunksjonalitet:** Testpersonen klarer oppgaven fint og har ingen innspill.

**Notifikasjon:** Testpersonen klarer oppgaven fint og har ingen innspill.

**Kampflyt:** Testpersonen finner ikke turneringstreet, og trenger hjelp til dette. Hun sier hun ikke vet hva en «Bracket» er. Resten av oppgavene får fint. Akkurat som brukertester 2 og 3, påpeker hun svakheten med «melde resultat»-modalen.

**Turneringsopprettelse:** Testeren klarer oppgaven fint. Hun skjønner hva formatet i turneringsfasene betyr, men synes siden ikke er så godt presentert. Hun vil også at vi bruker nedtrekksmeny istedenfor en komponent med pluss – og minusknapper, på «Advanced option» siden det ikke er tall, men ord. Hun trykker på en turneringsadministrator, som sender henne ut av registreringsskjemaet og hun mister alt hun har fylt inn. Ellers går registreringen fint.

**Generell tilbakemelding:** Hun er veldig alt i alt positiv til webappen og synes den er fin.

### 7.2.1.5 Brukertest 5

**Erfaringsnivå:** Testpersonen er en trent bruker og er medlem av spillgruppen i Accenture Norge. Han kjenner ikke særlig godt til prosjektet.

**Registrering:** Testpersonen klarer oppgaven fint og har ingen innspill

**Brukerprofil:** Testpersonen klarer oppgavene fint, men vil at modalen for å endre brukerinnstillingene skal lukke seg når man lagrer endringene. Han synes at laget skal komme opp med en gang på brukerprofilen etter opprettelse, akkurat som brukertester 4.

**Lagprofil:** Testpersonen klarer oppgavene fint og har ingen innspill.

**Turneringspåmelding:** Testpersonen klarer oppgavene fint, bortsett fra at han først trykker på en turnering med full påmelding. Han savner en tilbakeknapp i grensesnittet. Han trykker inn på turneringen istedenfor for å bruke «Join»-knappen fra turneringskortet. Han ønsker at «Join»-knappen på turneringsiden skal være større. I tillegg sier han at han ikke tenkte på «Join»-knappen fra turneringskortet. Dessuten sier han at gjerne vil vite hvilke turneringer lagene hans er med på med en visuell markering. Han foreslår en «My tournaments»-knapp, som viser turneringer hvor hans lags er med. Han ville også kunne søke på lag som er med turneringer på turneringsoversikten.

**Søkefunksjonalitet:** Testpersonen klarer oppgaven fint. Han ville gjerne ha lagene og turneringene når han søker på en bruker.

**Notifikasjon:** Testpersonen klarer oppgaven fint og har ingen innspill.

**Kampflyt:** Testpersonen klarer oppgavene fint. Akkurat som brukertester 2, 3 og 4, påpeker han svakheten med «melde resultat»-modalen.

**Turneringsopprettelse:** Testeren klarer oppgaven fint. Han mener språket på siden for formatet på turneringsformatet er upresist. Han mener «early rounds», «middle rounds» og «late rounds» er dårlig språk. Det er forståelig, men bør vi bør bruke noe som språk som gamere kjenner. Han vil gjerne også ha mulighet for å ha flere faser i en «Single elimination». Det vil si at man kan legge til en et nytt best-av-intervall i Figur 6-24. Dette kommer av at han velger en turnering med 512 lag. Tilsvarende mener han at om man har få antall lag som for eksempel åtte, kan det være greit å ha bare to turneringsfaser. Ellers går registreringen fint.

**Generell tilbakemelding:** Han er veldig alt i alt positiv til webappen og synes den er fin. Testpersonen utforsker webappen og påpeker at «zoomingen» på turneringstreet må ha en begrensning, slik at man unngår å ha et bittelitt turneringstre eller bare viser deler av en kamp. I tillegg prøver han å kontakte turneringsadministratører. Da får programmet en «Nullpointer exception».

## 7.2.2 Oppsummering av brukertestene

Det var positivt å høre at samtlige testpersoner var fornøyde med webappen i sin helhet.

Etter tilbakemeldingen fra brukertestene rettet vi følgende:

- Knappen for «Sing in» på landingssiden ble byttet til «Sign up». «Sign in» "knappen i navbaren forblir som før.
- Om en bruker skriver for kort passord får man beskjed om passordet må være minst åtte tegn
- Når en bruker lagrer endring av brukernavn/e-post på brukerprofilen lukkes modalen automatisk.
- Når en bruker skriver tittel på turneringsregistrering, kan vedkommende bruke tegnet «:».

- Når et kampresultat skal meldes kan brukeren skrive resultatet rett inn.
- Fikset «Nullpointer exception». Når en innlogget bruker kontakter turneringsadministrator(er) blir det sendt e-post til alle turneringsadministrator(er).

## 7.3 React Developer Tools

«React Developer Tool» er en utvidelse for nettleseren Google Chrome. Denne benyttet vi oss av hyppig under testing av klientkoden. Utvidelsen brukte vi blant annet til å manipulere de ulike komponentenes state for å raskt og enkelt kunne simulere ulike scenarier. Som for eksempel forskjell i brukergrensesnittet basert på om brukeren er eier av profilsiden den besøker eller ikke.

<https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadoplbjbfkapdkoienih>

## 7.4 Postman

Under oppsettet av de ulike endepunktene i APIet vårt benyttet vi oss av Postman for å enkelt kunne teste om de oppførte seg som ønsket. Postman tillot oss å utføre kall til endepunktene uavhengig om klientkoden som skulle benytte seg av disse var ferdig utviklet. Det var også en raskere måte å teste serversidens funksjonalitet enn å gjennomføre de ulike flyttene på klientsiden som for eksempel å fylle ut skjemaer.

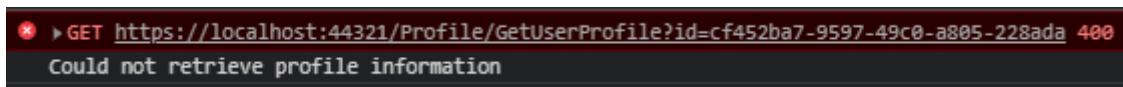
<https://www.postman.com/>

## 7.5 Chrome Devtools

Google Chrome kommer med en rekke utviklerverktøy direkte ut av boksen, som vi aktivt brukte gjennom hele utviklingsprosessen.

### 7.5.1 Konsollvinduet

For enkel testing og feilsøking av klientkoden så benyttet vi oss av konsollvinduet. Her blir det blant annet listet ut feilede serverkall. Det mest vanlige bruksområdet var å benytte JavaScript-kodelinjen «console.log()» for å skrive direkte til konsollen. Dette var spesielt nyttig for å få en oversikt over hvilken rekkefølge ulike deler av kode kjørte og hva slags verdier ulike variabler inneholdt på gitte tidspunkt i eksekveringen. Figuren under viser et eksempel på konsollvinduet etter å ha prøvd å aksessere en brukerprofil som ikke eksisterer.



```
* ▶ GET https://localhost:44321/Profile/GetUserProfile?id=cf452ba7-9597-49c0-a805-228ada 400
  Could not retrieve profile information
```

Figur 7-1 Konsollvinduet hvis bruker ikke eksisterer

### 7.5.2 Nettverksfanen

Nettverksfanen var et annet verktøy som hjalp oss mye med testing og feilsøking. Her kan man se en oversikt, i sanntid, over all kommunikasjon som blir gjort mellom klient og server. Oversikten inneholder en tabellvisning med ulik informasjon, om hver enkelt nettverkskommunikasjon. Det mest aktuelle for oss var statuskode, filtype, filstørrelse og tidsforbruk. Videre kan man velge en spesifikk nettverkskommunikasjon, og få utvidet informasjon om denne. Det vi hadde mest nytte av var «Status Code», «Request URL», «Payload» og «Response». «Status Code» refererer til standardiserte HTTP statuskoder serveren sender som del av et svar på en forespørsel fra klienten. 2xx betyr at alt gikk som det skulle, mens 4xx indikerer at serveren ikke kunne innfri forespørselen. «Request URL» består av tre deler. Første forteller hvilken nettverksprotokoll som blir benyttet, andre definerer hvilken server som blir kontaktet og tredje forteller hvilket endepunkt den kontaktede serveren skal videresende forespørselen til. «Payload» refererer til informasjonen som sendes sammen med forespørselen, og «Response» er informasjonen som serveren returnerer til klienten.

Her tar vi utgangspunkt i registrering av bruker som eksempel. Vi ser i nettverksfanen at klienten benytter nettverksprotokollen HTTPS for å sende en forespørsel til serveren «localhost» via port 44321 som videresender forespørselen til endepunktet «User/Register». Serveren responderer med statuskode 200 og med den genererte bruker-IDen, slik at koden videre kan videresende brukeren til den nyopprettede profilsiden sin. Vi ser også hva payloaden inneholder. I dette tilfellet er det spesielt viktig at HTTPS benyttes slik at payloaden, som inneholder brukeren sitt passord, blir kryptert før det sendes til serveren. Da forhindrer vi at eventuelle ondsinnet aktør som lytter til nettverkstrafikken, ikke får tilgang til passord eller annen data i klartekst. Hvis man prøver å registrere en ny bruker med samme e-post eller brukernavn vil serveren respondere med statuskode 400 og en melding med hva som forårsaket problemet.

```

General
Request URL: https://localhost:44321/User/Register
Request Method: POST
Status Code: 200
Remote Address: [::1]:44321
Referrer Policy: strict-origin-when-cross-origin

```

Figur 7-4 Request ULR og statuskode 200 (Success)

```

Request Payload view source
{
  "Email": "testbruker@test.no",
  "username": "Testbruker",
  "Password": "Tester123"
}
Email: "testbruker@test.no"
Password: "Tester123"
username: "Testbruker"

```

Figur 7-3 Payload med registreringsinformasjon

Response
"ab0d1bea-3e31-423f-94ca-4e28e4a16b10"

Figur 7-2 Respons med brukerID, etter vellykket registrering

```

General
Request URL: https://localhost:44321/User/Register
Request Method: POST
Status Code: 400
Remote Address: [::1]:44321
Referrer Policy: strict-origin-when-cross-origin

```

Figur 7-5 Statuskode 400 (Bad Request)

Response
This email address is not available

Figur 7-6 Respons etter feilet registrering med allerede registrert e-post

Response
This username is not available

Figur 7-7 Respons etter feilet registrering med allerede registrert brukernavn

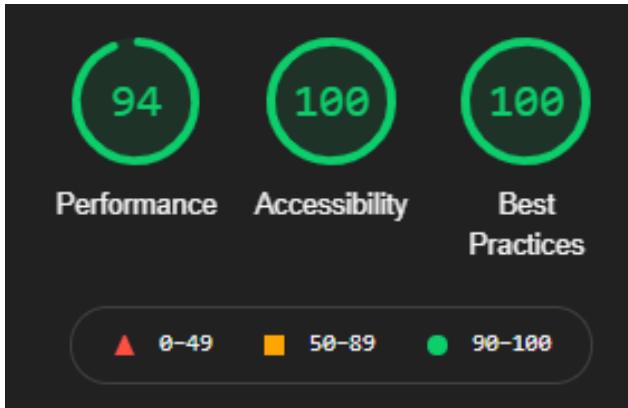
### 7.5.3 Lighthouse

Lighthouse er et verktøy som analyserer og bedømmer en nettside sin brukervennlighet i henhold til ytelse (Performance), universell utforming (Accessibility) og best praksis (Best Practices). Vi gjennomførte tester for både PC og mobil. Resultatene kan man se i figurene Figur 7-8 og Figur 7-9.

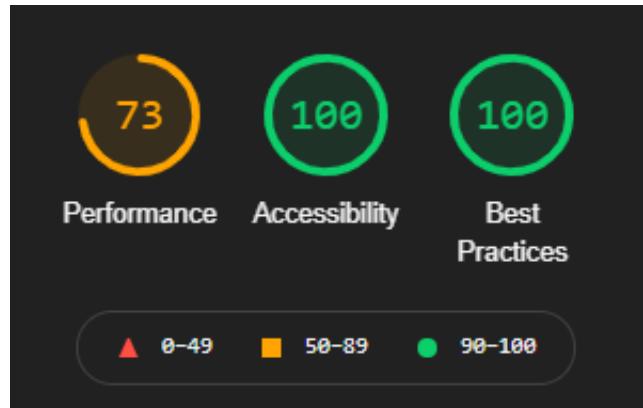
Ytelse refererer til hastigheten en nettside laster inn og er klar til bruk. Tester for mobil ga en del lavere uttelling for ytelse, men ettersom produkteier ikke mente mobiloptimalisering var av høy prioritet, brukte vi ikke unødvendig mye tid på å utbedre dette. For å forbedre dette i framtiden så er det mulig å i større grad komprimere applikasjonens bakgrunnsbilde og rense opp i ubrukte avhengigheter.

Universell utforming refererer til hvor godt tilpasset nettsiden er for å brukes av personer med ulike funksjonsnedsettelser, som for eksempel svaksynte eller blinde. Denne del av testen legger bland annet vekt på hvor godt nettsiden er strukturert for å tolkes av skjermlesere, og om det er tilstrekkelig kontrast mellom tekst og bakgrunn. Vi prioriterte dette høyt ettersom nettsider er pålagt å møte et minstekrav når det kommer til universell utforming jf. § 4 i Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger.

Best praksis refererer blant annet til hvor godt nettsiden følger standarder når det kommer til optimalisering og sikkerhet. Noe som regnes som dårlig praksis er hvis nettsiden gjør nettverkskall til kilder som er flagget som usikre eller ikke benytter HTTPS-protokollen.



Figur 7-8 Lighthouse-analyse av nettsiden (PC)



Figur 7-9 Lighthouse-analyse av nettsiden (mobil)

## 7.6 Azure app service

Det å teste webappen i «produksjon» er viktig. Vi opprettet flere Azure app services slik at alle gruppemedlemmene kunne teste løsningen i «produksjon». Vi oppdaget problemene med database, bildeopplastning ved å publisere til Azure app service.(Se kapittel 6.5.5. Dessuten måtte vi bruke Azure app service for å kunne teste Steam og Discord.

# **8 Brukermanual**

## **8.1 Forord**

I dette kapittelet skal vi samle alle ressurser en uerfaren bruker trenger for å kunne kjøre løsningen lokalt. I første del forklarer statuskodene vi har laget for turneringer og kamper. Vi vil liste opp alle programmer som trengs og vise til ressurser for å kjøre programmet. Til slutt vil vi belyse mangler, eventuelle feil og videre arbeid for team som skal jobbe med systemet videre.

## **8.2 Statuskoder**

### **8.2.1 Turnering**

0 = Påmelding er åpen

1 = Påmelding er lukket

2 = Turneringen har startet

3 = Turneringen er fullført

### **8.2.2 Kamp**

0 = Kampen har ikke startet

1 = Kampen har startet

2 = Kampen er ferdigspilt, resultat er satt

## 8.3 Hvordan kjøre webapplikasjonen lokalt

### 8.3.1 Programmer

Program	Bruksområde
Node.js	Helt nødvendig for at React skal kunne kjøre
Visual Studio	Dette er IDEen som er brukt i utviklingen og gjør det lett å kjøre opp systemet
Ekstra: Git DB Browser for SQLite	Man kan da klone prosjektet om man har tilgang til repositoriet i Github.  Med dette programmet kan man sjekke endringer og vedlikeholde databasen.

### 8.3.2 Ressurser

Nedlastning/kjøring	Windows	Mac
Node.js nedlastning	<a href="https://nodesource.com/blog/installing-nodejs-tutorial-windows/">https://nodesource.com/blog/installing-nodejs-tutorial-windows/</a>	<a href="https://nodesource.com/blog/installing-nodejs-tutorial-mac-os-x/">https://nodesource.com/blog/installing-nodejs-tutorial-mac-os-x/</a>

Visual studio nedlastning	<a href="https://www.guru99.com/download-install-visual-studio.html">https://www.guru99.com/download-install-visual-studio.html</a>	<a href="https://tutorials.visualstudio.com/vs4mac-install/install">https://tutorials.visualstudio.com/vs4mac-install/install</a>
Kjøring	<ul style="list-style-type: none"> <li>-Pakk ut koden</li> <li>-Åpne Visual studio</li> <li>-Velg «Open project»</li> <li>- Velg SLN-filen</li> <li>-Trykk «Run»-knapp</li> </ul> <p>(Må vente en stund til node modulene er installert)</p>	<ul style="list-style-type: none"> <li>-Pakk ut koden</li> <li>-Åpne Visual studio</li> <li>-Velg «Open project»</li> <li>- Velg SLN-filen</li> <li>- Trykk «Run»-knapp</li> </ul> <p>(Må vente en stund til node modulene er installert)</p>

## 8.4 Mangler og videre arbeid

### 8.4.1 Mangler

- Slette lag
- Discord-integrasjon
- På brukerprofilen har man ubegrenset antall forsøk på å endre passordet
- Når man oppretter en turnering eller endrer en turnering og vil invitere turneringsadministratorer, men trykker på deres profil, vil man miste alle endringer som er fylt inn.
- Turneringer tar ikke hensyn til forskjellige tidssoner

## 8.4.2 Videre arbeid

I dette avsnittet lister vi opp utvidelser vi ser for oss basert på brukertester, egne erfaringer med eksisterende turneringsportaler og idemyldring gjort i starten av prosjektperioden.

På kort sikt:

- Fikse manglene beskrevet i 8.4.1.
- Tooltips på knapper
- Laste opp bilder i chat
- Lagre turneringsoppsett til pdf eller bilde.
- Tidspunkt på kamper i turneringen
- Legge til flere formater:
  - single elimination med gruppespill
  - seriespill
  - double elimination (eventuelt med gruppespill)
- Legge til flere spillkontoer:
  - Riot
  - Faceit
  - Blizzard
  - EA profil
- Neste kamp på lagprofilen
- Mine turneringer på turneringsoversikten
- Se gamle, pågående og nye turneringer på turneringsoversikten
- Ved søk på lag, turnering eller brukere skal man se tilknytninger (For eksempel: Man kan se lagene en bruker er med på)

På lengre sikt:

- Integrere APIer for spillstatistikk
- Lage klubber

## 9 Ordliste

Serverside: Delen av applikasjonen som kjører på serveren

Serverkode: Koden som kjører på serversiden

Klientside: Delen av applikasjonen som kjører på klienten

Klientkode: Koden som kjører på klientsiden

Render: Gjengir elementer i brukergrensesnittet til brukeren

Merge conflicts: Konsept i samarbeidsverktøy for kode som går ut på at man vil merge(flette) kode som er ulik i samme fil.

Modal: En boks med innhold som legger seg foran resten av innholdet fremfor å ta brukerne til en ny side med innhold.

Cookie: En informasjonskapsel som lagrer informasjon om brukeren til ulike formål

API: Application Programming Interface er et grensesnitt på serversiden som gjør det mulig for ulike klientsider å benytte seg av samme serverkoden gjennom serverkall til endepunkter.

Endepunkt: Delen av APIet som håndterer de ulike serverkallene.

Single elimination: Et turneringsformat med vanlige utslagsrunder.

## 10 Referanser

*React – A JavaScript library for building user interfaces.* – A JavaScript library for building user interfaces. <https://reactjs.org/>.

*MVC Framework - Introduction.* Tutorialspoint.

[https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm).

Fitzmac, VikasPullagura-MSFT, davidsmatlak et al. *Azure Resource Manager overview - Azure Resource Manager*. Azure Resource Manager overview - Azure Resource Manager | Microsoft Docs. <https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview>.

# 11 Vedlegg

## 11.1 Oppdragsgivers vurdering

**Oslo 21.05.2021** Til den det måtte angå

**Prosjekt:** Accenture e-sport

**Innoverte:** Daniel Roger Hansen, Herman Vognild Rustad, Paul Mathias Høglend

**Prosjektperiode:** 12.01.2021 - 03.06.2021

**Veilederes vurdering:**

Gjennom våren har gruppen utviklet en turneringsplattform som skal tas i bruk av spill entusiaster i Accenture både nasjonalt og internasjonalt. Plattformen legger til rette for opprettelse, oppfølging og gjennomføring av turneringer på en mye enklere måte enn eksisterende manuelle prosesser har gjort.

Helt fra prosjektets startfase har gruppen tatt eierskap til produktet de skal levere og har med stolthet vist frem arbeidet sitt etter hvert som deler av systemet har blitt ferdigstilt. De har jobbet veldig selvstendig, samtidig som de ikke har vært redde for å be om oppklaringer eller vurderinger fra veiledere og produkteier der de har sett det nødvendig. Hvert enkelt gruppemedlem har vist stort engasjement og gode tekniske ferdigheter.

Gjennom prosjektperioden har produkteier gjort enkelte større endringer i produktkøen etter hvert som behov har endret seg. Dette har gruppen håndtert meget godt, noe som viser til deres store grad av tilpasningsdyktighet.

Alt i alt er vi veldig fornøyde med det enkeltpersonene og gruppen som en helhet har produsert og levert gjennom deres bachelorprosjekt.

Med vennlig hilsen,

Elisabeth B. Karud, veileder

Tlf: +47 988 19 641

E-post: [elisabeth.b.karud@accenture.com](mailto:elisabeth.b.karud@accenture.com)

Jens Omfjord, veileder

Tlf: +47 986 73 111

E-post: [jens.omfjord@accenture.com](mailto:jens.omfjord@accenture.com)

Sindre Røkke, produkteier

Tlf: +47 958 50 334

E-post: [sindre.rokke@accenture.com](mailto:sindre.rokke@accenture.com)

## 11.2 Kravspesifikasjon

### 11.2.1 Bakgrunn

Siden august 2020 har Accenture Norge hatt et veldig lite gamingmiljø som har blitt administrert gjennom e-post og Teams. Denne gruppen heter "Spillgruppen" og er en altomfattende gruppe som dekker både kompetitiv gaming, avslappet gaming, brettspill og mye mer.

I august (2020) ble det undersøkt om det var interesse internt i selskapet for å samle folk til et lag og melde seg på en bedriftsliga i spillet Counter-Strike: Global Offensive (CS: GO). Undersøkelsen viste at det var en stor interesse og 20 ansatte ble med på dette.

Det har blitt gjennomført interne turneringer før, men disse har vært små og blitt administrert manuelt gjennom Excel. Om det skal bli arrangert noe større så blir det store mengder manuelt arbeid, og man må vurdere å leie inn noen for å ordne dette.

### 11.2.2 Leserveiledning

Denne kravspesifikasjonen er ment for utviklere som skal utvikle plattformen og spillgruppen i Accenture Norge for å sørge for at produktet som utvikles er i henhold til avtalen mellom partene. Dette dokumentet går igjennom: dagens situasjon, avtalt prosjektomfang for et bachelorprosjekt, funksjonelle, ikke-funksjonelle krav og litt om rammebetingelsene.

### 11.2.3 Prosjektomfang

Vi ble presentert for følgende oppgavetekst:

**App for e-sport gruppen**

Applikasjon for Accenture nystartede e-sportslag i CS. En applikasjon hvor man kan opprette turneringer for spillere og bruke APIer for å hente ut spiller statistikk.

Adminfunksjonalitet:

- Opprette lag og legge til spillere
- Sende ut varsel til spillere
- Lage turneringer

Medlemsfunksjonalitet:

- Følge opp statistikk
- Delta i turneringer
- Profilside

Gruppen justerte oppgaven i henhold til oppgaveteksten med en gang. I samråd med veiledere og produkteier ble vi enige om å lage en turneringsplattform for flere typer dataspill. I tillegg ville vi ikke inkludere eksterne APIer for spillstatistikk, da dette ville bli alt for omfattende for prosjektet. Til slutt kom vi fram til at oppgaven skulle være:

Oppgaven går ut på å designe og utvikle en turneringsplattform der man kan registrere seg som bruker, bli med på lag og delta i turneringer i forskjellige dataspill. I tillegg skal man ha mulighet til å opprette turneringer.

## 11.2.4 Funksjonelle krav

### 11.2.4.1 Registrering og innlogging

Nye brukere skal kunne registrere seg og logg inn på plattformen. Brukerstøtte for gjenopprettning av passord skal også implementeres.

#### **11.2.4.2 Brukerprofil**

Brukerprofilen skal inneholde blant annet informasjon om en enkeltspiller og en oversikt over alle lagene man spiller for. I tillegg skal det være mulig å knytte opp profilen sin på plattformen til andre organisasjoner spillerprofiler. Det skal også være mulig å bytte profilbilde og banner.

#### **11.2.4.3 Lag**

Alle brukere skal ha muligheten til å registrere et nytt lag. Lagene skal ha en profil med oversikt over alle brukerne som er medlem, og alle tidligere kamper laget har spilt. Det skal også være en lagleder som har mulighet til å invitere spillere til laget, fjerne spillere og bytte profilbilde og banner på profilsiden. Laglederen skal også ha muligheten til å melde laget på turneringer.

#### **11.2.4.4 Turneringer**

Innloggede brukere, skal ha mulighet til å opprette turneringer. Ved oppretting av en ny turnering blir turneringsformat og en rekke andre parametere satt. Basert på formatet, parametere og antall påmeldte lag skal systemet automatisk generere et turneringstre eller kampoppsett. Plattformen skal ha en oversikt over tidligere, pågående og kommende turneringer, samt en side med informasjon for hver enkelt turnering. Alle kamper i en turnering skal ha et kampprom der det står hvilke lag som møtes. Her skal det være en privat chat for alle deltagere i kampen og en administrator for tildeling av viktig informasjon knyttet til kampen.

Oppdatering: En turneringsoversikt skal kunne filtrere på land.

#### **11.2.4.5 Varsel**

Det skal være mulig å få varsler som innlogget bruker. Det er hovedsakelig varsler fra brukeren som organisere en turnering man er meldt på, eller varsle om at man har fått en melding i et kampprom man er med i.

#### **11.2.4.6 Søkefunksjonalitet**

Brukere skal kunne søke opp andre brukere, lag og turneringer.

#### **11.2.4.7 Administrator**

En del av systemet skal bare være tilgjengelig for brukere som har blitt tildelt administratorrettigheter. Her skal det implementeres nødvendige oversikter og funksjonalitet for å kunne vedlikeholde systemet.

#### **11.2.4.8 Autentisering og sikkerhet**

Sessions for å sjekke om bruker er innlogget og for å validere brukeren sin rolle i systemet. Informasjonskapsler (Cookies) for å holde brukeren innlogget.

### **11.2.5 Ikke-funksjonelle krav**

#### **11.2.5.1 Sikker lagring av passord**

Passord skal lagres kryptert i databasen ved hjelp av en hashingalgoritme

#### **11.2.5.2 Passordstyrke**

Passordene skal være minst åtte tegn med minst et tall og en stor bokstav

#### **11.2.5.3 Responsivt design**

Webappen skal skalere til mindre skjermer.

#### **11.2.5.4 Universell utforming**

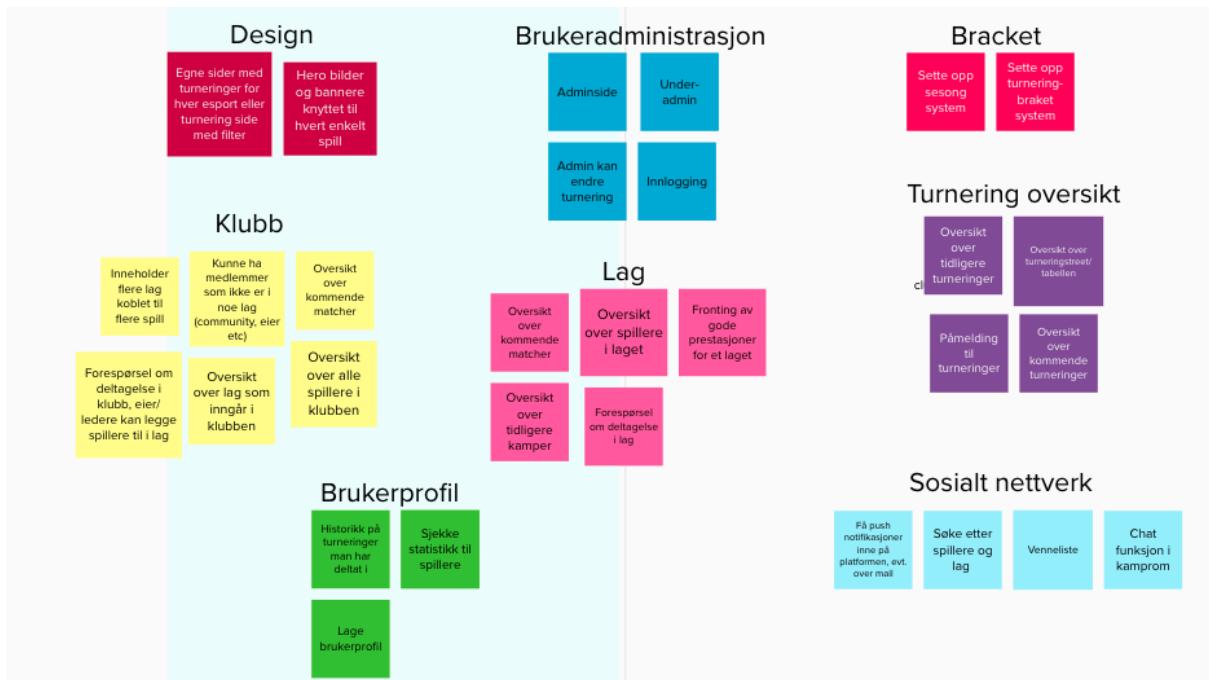
Nettløsninger skal minst utformes i samsvar med standard Web Content Accessibility Guidelines 2.0 (WCAG 2.0)/NS/ISO/IEC 40500:2012, på nivå A og AA med unntak for suksesskriteriene 1.2.3, 1.2.4 og 1.2.5, eller tilsvarende denne standard. Jf. § 4 i Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger

#### **11.2.6 Rammebetingelser**

Løsningen skal utvikles som en webapp. Accenture står som eier av produktet. Gruppen skal benytte en smidig prosessmodell. *Språket i brukergrensesnittet skal være norsk.* Prosjektet skal være ferdig til 26. mai. Gruppen skal bruke teknologiene .NET og React.

*Oppdatering: I mars 2021 kunne produkteier fortelle at systemet skal tas i bruk etter endt bachelorperiode og vil følgelig bli videreutviklet av et eget team. Han ba også om at språket i brukergrensesnittet endres til engelsk slik at Accenture i andre land også benytte seg av plattformen.*

## 11.3 Design thinking iterasjon 2



Figur 11-1 Iterasjon to av oppgavene brainstorm og clustering

	Brukertilgang	Adminside	Lag	Legge til turneringer	
Avhengigheter	Brukerprofil Registrering Sikkerhet Rolle: Bruker	Adminside Rolle: Admin	Brukertilgang Invitasjons-system Klubbregistrering Klubboversikt	Lag Rolle: Lagleder Klubboversikt Hjemmeside som skal inkluderes Velg av turneringstema	Legge til turneringer Organisator
Tekniske utfordringer	Brukertilgang Bruker profil med mulighet for å sette opp et team, med egen tilgangskode	Håndtering av sessions Skilte på ulike roller	Klubb oversikt Klubboversikt med lag Klubboversikt for å finne medlemmer		
Tid	15	18	15	30	10 12 54%

Figur 11-2 Iterasjon to av oppgaven creative matrix



Figur 11-3 Iterasjon to av oppgaven *Importance/difficulty matrix*

Klubb var en ambisiøs utvidelse etter modell fra <https://gamer.no>. Vi hadde ikke tid til å implementere dette.