

Högskolan i Gävle

# Sportsapplikation

---

*Linus Renström*

[linus.renstrom@icloud.com](mailto:linus.renstrom@icloud.com)

*Emre Özata*

[emre10909@gmail.com](mailto:emre10909@gmail.com)

*Brandon Dalima Nkata*

[brandondalima@gmail.com](mailto:brandondalima@gmail.com)

2025-01-14

Kurs: Klientutveckling på mobila enheter

---

Lärare: Åke Wallin

Lärare/handledare: Åke Wallin

# Innehållsförteckning

1	Inledning .....	1
2	Metod.....	2
2.1	Planering.....	2
2.2	Projektstart och Konfiguration .....	2
2.3	Dataklasser och anrop till API .....	2
2.4	Fragment .....	3
2.5	Visa lag, ställning, poäng, skytteliga och matcher .....	4
2.6	Stöd för olika språk .....	4
2.7	RecyclerView .....	4
2.8	Databas.....	5
2.9	Shared preferences .....	5
3	Resultat.....	6
3.1	Visa sportdata .....	6
3.2	Chattrum och firebase-databas.....	12
3.3	UML-diagram.....	17
4	Analys och diskussionen .....	18

# **1 Inledning**

Till detta arbete har vi skapat en sportsapplikation där vi har tagit inspiration från riktiga applikationer. Denna applikation har det fokuserats på att skapa en robust applikation där vi har tagit till beaktning att använda oss av clean code. Vi har också lagt ett stort fokus kring att urskilja mellan logik och de grafiska komponenterna för att uppnå en bra separation.

## 2 Metod

Här kommer det att presenteras kring tillvägagångssättet vid skapandet av applikationen.

### 2.1 Planering

Till en början arbetade vi i projektgruppen tillsammans för att skapa en skiss över hur vi ville att applikationen skulle fungera. Vi ritade ut hur applikationen skulle navigeras och vilka de viktigaste funktionerna var. Vi bestämde oss för att använda olika fragment med en underliggande aktivitet. Fragmenten skulle vara generella och anpassningsbara, att de oavsett sport kan visa liknande data från det hämtade API:et.

### 2.2 Projektstart och Konfiguration

Sedan skapade vi en ny, tom "views activity" och konfigurerade Git att alla medlemmar kunde komma åt projektet. För att kommunicera mot API:et använde vi Retrofit.

### 2.3 Dataklasser och anrop till API

Därefter skapade vi en stor dataklass för att kunna objektifiera svaren vi fick från API:et. Vi använde GSON konverter för detta, och det var viktigt att våra dataklasser matchade JSON-formatet och svaret vi fick från API:et. Med hjälp av dataklasserna kunde vi sedan sortera och filtrera datan för att till exempel kunna skapa ett gränssnitt där lagen i serien visas i rätt ordning.

Vi skapade också en SportsAPIService för att kunna göra olika anrop till API:et. Detta gör att vi med hjälp av anrop kan hämta information direkt från API:et och sedan visa informationen i vår applikation. Varje anrop kräver en ny metod för att hämta korrekt data, där vi definierar parametrar från den sparade informationen i ViewModel.

Till en början hade vi problem med kommunikationen med det givna API:et, då vi tidigare endast hade använt ett API utan en nyckel. Genom flera försök och sökningar på internet lyckades vi till slut kommunicera med API:et. För att testa kommunikationen skapade vi ett enkelt gränssnitt för att förstå hur allt hängde ihop.

## 2.4 Fragment

När vi väl hade förstått hur kommunikationen fungerade skapade vi ett fragment som användaren välkomnas med. Till en början visade fragmentet en enkel välkomsttext, men efter diskussion inom gruppen har vi skapat en animation (se figur 1). I den underliggande aktiviteten har vi skapat en fragmentbehållare för att visa de olika fragmenten samt en navigeringsmeny ("navigation drawer") (se figur 2). Navigeringsmenyn implementerades genom att hämta alla tillgängliga sporter från API:et och sedan presentera dem. När användaren klickar på en sport navigeras de till nästa fragment. Den valda sporten sparas med hjälp av underliggande kod, och denna information används för att visa rätt information i nästa fragment. Denna data lagras i våra skapade ViewModel-klasser för att kunna behålla och hämta information mellan de olika fragmenten.

När vi navigerar från WelcomeFragment används en "navigation controller" för att navigera till ShowLeagues-fragmentet (se figur 3). "Action" som beskriver navigeringen definierades i en navigeringsgraf. Navigeringen sker utan att något argument skickas med. Men eftersom ligorna som visas senare ska anpassas till vald sport måste sportens ID sparas och skickas till nästa fragment. Det valda sportens ID sparas i en LiveData-attribut i LeaguesViewModel. ShowLeagues-fragmentet kan begära en referens till LeaguesViewModel med hjälp av klassen ViewModelProvider. Via den referensen kan ShowLeagues-fragmentet lyssna på attributet och visa ligor med hänsyn till ID-attributet. Samma princip används vid navigering från showLeagues till showTeams

## 2.5 Visa lag, ställning, poäng, skytteliga och matcher

Som tidigare nämnt visas ligorna med hjälp av information från API:et. När användaren trycker på en liga kommer navigeras användaren till ett nytt fragment som heter `fragment_show_teams`, som består av fem slides (se figur 4): "Show teams" (visar lagen i ligan), "show standings" (visar ligatabellen), "show scoring-leaders" (visar de som har gjort flest mål), "top point-leaders" (visar spelare med flest poäng) och "show matches" (visar matcher). Varje slide har en egen observer som uppdaterar vyn i fragmentet (se figur 5–11). Eftersom vi har sparat ligans ID i vår ViewModel vet vi vilken liga vi ska hämta resterande data ifrån. När en slide klickas rensas vyn och den nya vyn visas. Alla slides har en egen metod som gör ett anrop till API:et för att hämta rätt data. Exempelvis har "show\_standings" metoden `handleStandingsTab()` och "top point-leaders" metoden `handleScoringLeadersTab()`. Show matches har också en funktionalitet där användaren kan sortera datat på kommande matcher och redan spelade matcher. Tidigare hade vi implementerat att det gick att filtrera på matcher som är live men eftersom vårt API inte tillhandhåller denna data tog vi bort denna funktion.

## 2.6 Stöd för olika språk

Vi har också implementerat stöd för flera språk i applikationen. Detta görs genom att skapa resursfiler för de olika språken, som sedan anpassas beroende på användarens språkställningar.

Till en början använde vi `findViewById` men efter projektets gång övervägde vi till att använda oss av `ViewBinding`. Genom denna ändring medför det till att vår kod blev alltmer typsäkert där det nu inte kastar exceptions om vi exempelvis anger en felaktig id till en view komponent.

## 2.7 RecyclerView

För att utveckla vårt GUI och göra det snyggare valde vi också att övergå från de vanliga vyerna likt `LinearLayout` och `ScrollView` till att använda oss av `RecyclerView` (se figur 3–11). Genom detta övervägande medförde det till ett mer unikt gränssnitt som vi anser skulle locka slutanvändaren mer. Det medför också till att vår kod blev mer separerad än tidigare som i sin tur ökar förståelsen kring koden.

Det blev dessutom lättare att anpassa designen på vyerna då data sparades i form av dataklasser. Objekt av dessa klasser presenterades med en egen vy, kallade `CardViews` som dyker upp i `RecyclerView`. Dessa `CardViews` kopplades till en `RecyclerView` i en fragments xml-fil med hjälp av adapterklass. En adapterklass skrevs för varje `RecyclerView` som skapades i fragment en layout.

## 2.8 Databas

Till vårt projekt tänkte vi ha en chattfunktion där meddelanden sparas och struktureras i Firebase real time database. Detta utfördes genom att skapa ett messagefragment med en uppkoppling till databasen (se figur 12–13). Detta chattrum ska vara unikt beroende på vilken liga användaren är inne på. Därför måste applikationen hålla koll på vilken liga användaren är inne på genom liga namnet. Detta liga namn sparas i en LiveData som vi observerar och sedan skickas liga namnet upp som en barnnod till subträdet som heter *Leagues* som är ett subträd till *Messages*. Detta görs med funktioner som `push()` som skapar data i databasen och `child()` som skapar en barn nod till en befintlig nod. I vår message nod under ligan i Firebase presenteras även vilka members som är med (se figur 14). För att skapa mer förståelse kring vilket chattrum som användaren befinner sig inom har vi också implementerat att det tydligt syns kring vilket chattrum som användaren befinner sig inom.

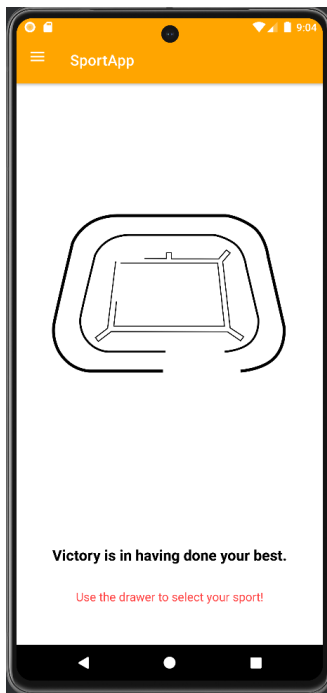
## 2.9 Shared preferences

I vår chattfunktion hade vi tidigare implementationen där användaren skrev in ett användarnamn varje gång personen skulle skicka meddelande. Denna design var inte användarvänlig utan i stället blev det mer användarvänligt om där det endast behöver ange ett användarnamn endast en gång som sparas när under applikationens livstid samt som sparas varje gång användaren startar om appen. Detta löstes med Shared Preferences som kan lagra små data lokalt på en android enhet. Detta gör att användarnamnet användaren matat in kommer att sparas ända tills appen av installerats. Vi har även ordnat att det är möjligt att byta användarnamn vilket tar bort det gamla och ersätter det med det nya användarnamnet (se figur 15–17).

## 3 Resultat

### 3.1 Visa sportdata

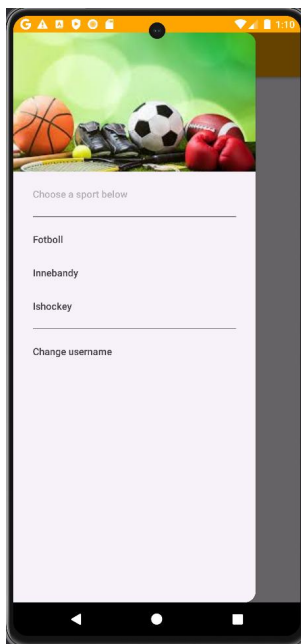
När applikationen startas visas följande fragment "fragment\_welcome" (se figur 1). Som ett designval har en animation lagts till fragmentets layout. Animationen fungerar tillsammans med de varierande citaten för att ge denna sida på applikationen en mer livlig upplevelse för användarna.



*Figur 1 - bilden visar fragmentet som visas när applikationen startas.*

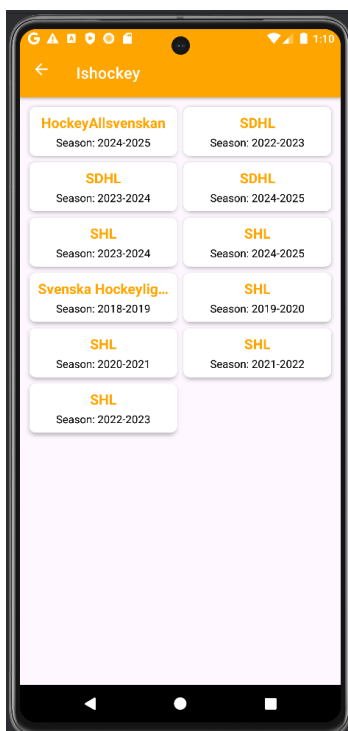


För att kunna navigera vidare måste användaren trycka på navigation drawern längst upp i vänstra hörnet. Då visas följande (se figur 2).



*Figur 2 - bilden visar navigation drawer där användaren kan välja sport eller att byta sitt användarnamn (mer om användarnamn senare).*

Sedan måste en sport väljas under rubriken ”choose a sport”. Fragmentet uppdateras lika och visar data beroende på vilken sport användaren valt. I detta visningsexempel valdes ishockey. Detta gav följande (se figur 3).



*Figur 3 - bilden visar ligor som tillhör den valda sporten.*

Om exempelvis fotboll valts hade fotbollsligor som allsvenskan och superettan visats. Efter detta fragment "fragment\_show\_leagues" navigerades det vidare till *SHL* – *Season 2024–2025*, detta gav följande (se figur 4). Här inne finns det 5 flikar som är klickbara. Om "show teams" markeras visas följande (se figur 4).



Figur 4 - bilden visar lagen till den tillhörande ligan.

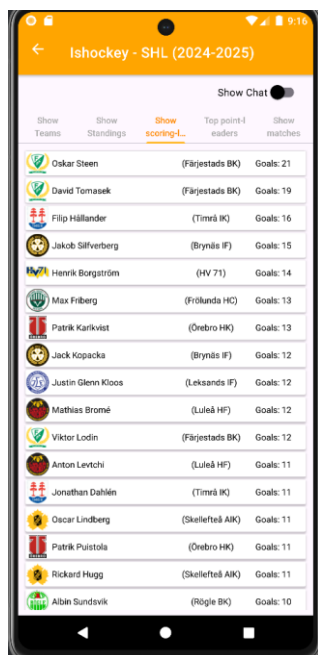
Om markerar trycker på "show standings" visas följande (se figur 5).

The screenshot shows the 'Ishockey - SHL (2024-2025)' app interface with the 'Show Standings' tab selected. The table lists 12 teams with their positions, logos, and statistics. The statistics include Played, Wins, Draws, Losses, and Points.

Position	Team	Played	Wins	Draws	Losses	Points
1	Brynäs IF	31	15	9	7	61
2	Färjestads BK	31	15	9	7	59
3	Frölunda HC	31	15	5	11	55
4	Luleå HF	31	13	8	10	54
5	Timrå IK	31	14	6	11	52
6	Örebro HK	31	11	11	9	48
7	Skellefteå AIF	31	14	4	13	48
8	IF Malmö Redhawks	31	10	9	12	43
9	Rögle BK	31	10	8	13	42
10	Växjö Lakers HC	31	10	9	12	42
11	Leksands IF	31	10	9	12	41
12	HV 71	31	8	10	13	38
	Linköping HC	31	8	9	14	37

Figur 5 - bilden visar liga-tabellen tillsammans med annan information till den tillhörande ligan.

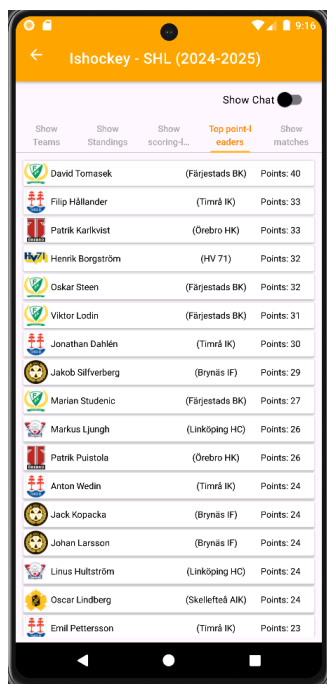
Om navigerar ”show scoring leaders” visas följande (se figur 6).



Ishockey - SHL (2024-2025)		
Show Chat		
Show Teams	Show Standings	Show scoring leaders
Oskar Steen	(Färjestads BK)	Goals: 21
David Tomasek	(Färjestads BK)	Goals: 19
Filip Hållander	(Timrå IK)	Goals: 16
Jakob Silfverberg	(Brynäs IF)	Goals: 15
Henrik Borgström	(HV 71)	Goals: 14
Max Friberg	(Frölunda HC)	Goals: 13
Patrik Karlqvist	(Örebro HK)	Goals: 13
Jack Kopacka	(Brynäs IF)	Goals: 12
Justin Glenn Kloos	(Leksands IF)	Goals: 12
Mathias Bromé	(Luleå HF)	Goals: 12
Viktor Lodin	(Färjestads BK)	Goals: 12
Anton Levchi	(Luleå HF)	Goals: 11
Jonathan Dahlén	(Timrå IK)	Goals: 11
Oscar Lindberg	(Skeleffä AIK)	Goals: 11
Patrik Puustola	(Örebro HK)	Goals: 11
Rickard Hugg	(Skeleffä AIK)	Goals: 11
Albin Sundsvik	(Rögle BK)	Goals: 10

Figur 6 - bilden visar skytteligan till den tillhörande ligan.

Efter detta navigerades det till top point leaders (se figur 7).



Ishockey - SHL (2024-2025)		
Show Chat		
Show Teams	Show Standings	Show Top point leaders
David Tomasek	(Färjestads BK)	Points: 40
Filip Hållander	(Timrå IK)	Points: 33
Patrik Karlqvist	(Örebro HK)	Points: 33
Henrik Borgström	(HV 71)	Points: 32
Oskar Steen	(Färjestads BK)	Points: 32
Viktor Lodin	(Färjestads BK)	Points: 31
Jonathan Dahlén	(Timrå IK)	Points: 30
Jakob Silfverberg	(Brynäs IF)	Points: 29
Marian Studenic	(Färjestads BK)	Points: 27
Markus Ljungh	(Linköping HC)	Points: 26
Patrik Puustola	(Örebro HK)	Points: 26
Anton Wedin	(Timrå IK)	Points: 24
Jack Kopacka	(Brynäs IF)	Points: 24
Johan Larsson	(Brynäs IF)	Points: 24
Linus Hultström	(Linköping HC)	Points: 24
Oscar Lindberg	(Skeleffä AIK)	Points: 24
Emil Pettersson	(Timrå IK)	Points: 23

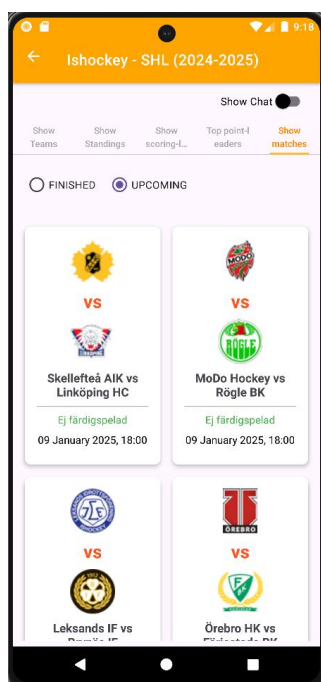
Figur 7 - bilden visar poäng-ligan till den tillhörande ligan.

Efter detta navigerades det till *show matches* där användaren kommer få ett val att se pågående matcher eller avslarade matcher (se figur 8).



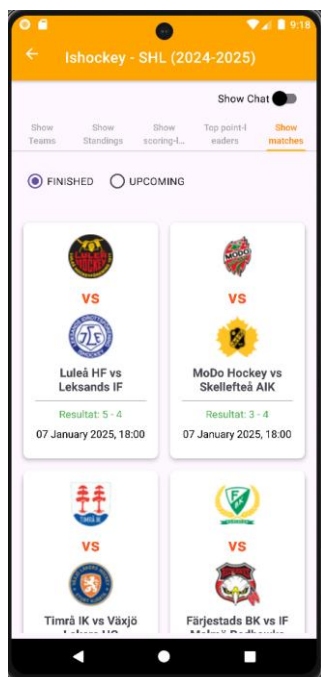
Figur 8 - bilden visar alternativet om de vill se kommande matcher eller avslarade matcher.

Först navigeras det till uppkommande matcher (se figur 9).



Figur 9 - bilden visar uppkommande matcher till den tillhörande ligan.

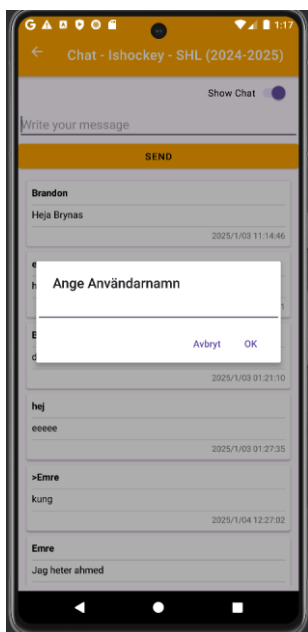
Sedan till avklarade matcher (se figur 10).



Figur 10 - bilden visar avklarade matcher till den tillhörande ligan.

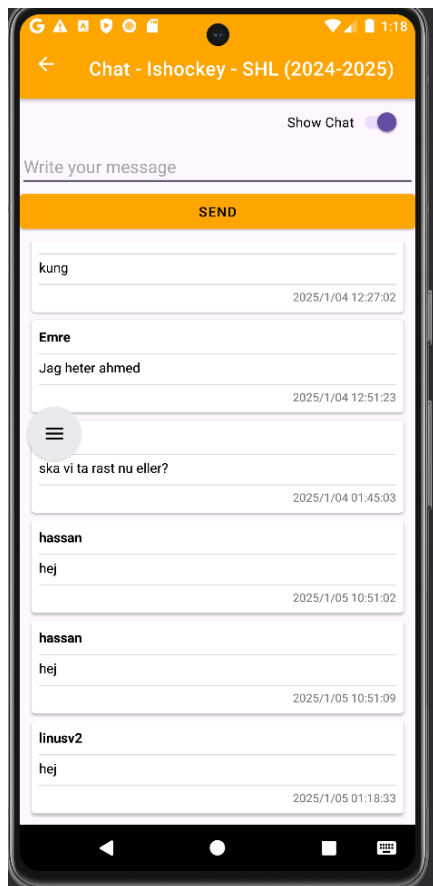
### 3.2 Chattrum och firebase-databas

Då användaren valt en liga finns en switch uppe i högra hörnet där det står *Show Chat*. Om denna trycks kommer det visas ett fragment som agerar som chattfunktion. Denna chatt tillhör den specifikt valda ligan som användaren är inne på just nu och chattar sparas i en firebase realtime database. Då switchen "showchat" markeras visas följande (se figur 11).



*Figur 11 - bilden visar chatt-rummet som är unikt för varje liga.*

För att kunna chatta måste du ange ett användarnamn. Om du väljer *Avbryt* kommer du få se chattrummet. Om användaren väljer att skicka ett meddelande utan användarnamn kommer samma dialogruta upp igen som tidigare. I denna exempelvisning valdes namnet *linusv2* och sedan skickades en chatt, vilket gav följande (se figur 12).



Figur 12 - bilden visar chatt-rummet efter ett meddelande har skickats.

Som figuren visar skickades denna i *SHL (2024–2025)* chatten, vilket kommer lägga till följande i firebase konsolen (se figur 13)

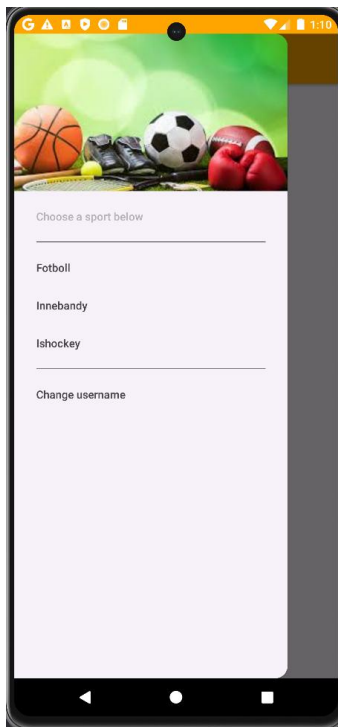


Figur 13 - bilden visar databasen som är ihopkopplat till chattrummet.

I trädet kan de även se att ligorna är strukturerade och har ett eget chattrum.

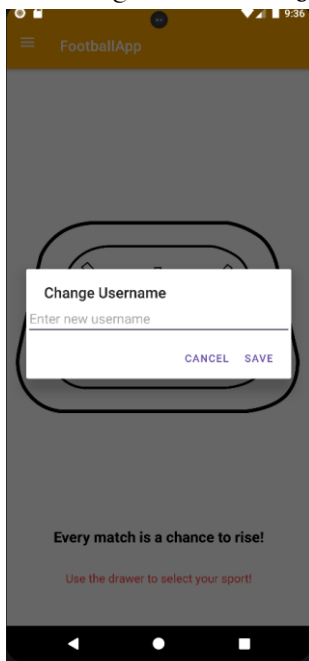


Då det har angivits ett användarnamn kan användaren även byta användarnamn i navigation drawern -> change username (se figur 14).



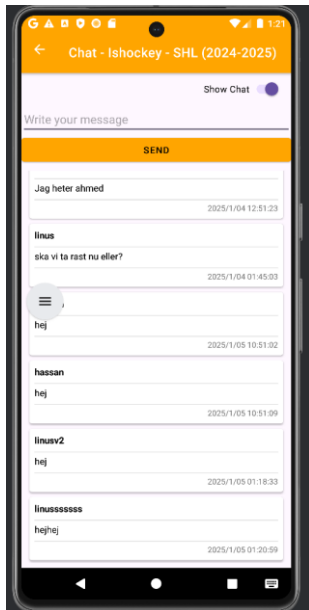
Figur 14 - bilden visar navigation drawer och under sporter byter man användarnamn.

Här navigeras till "Change username" och då dyker följande ruta upp (se figur 15).



Figur 15 - bilden visar dialogrutan som kommer upp om användaren vill byta användarnamn.

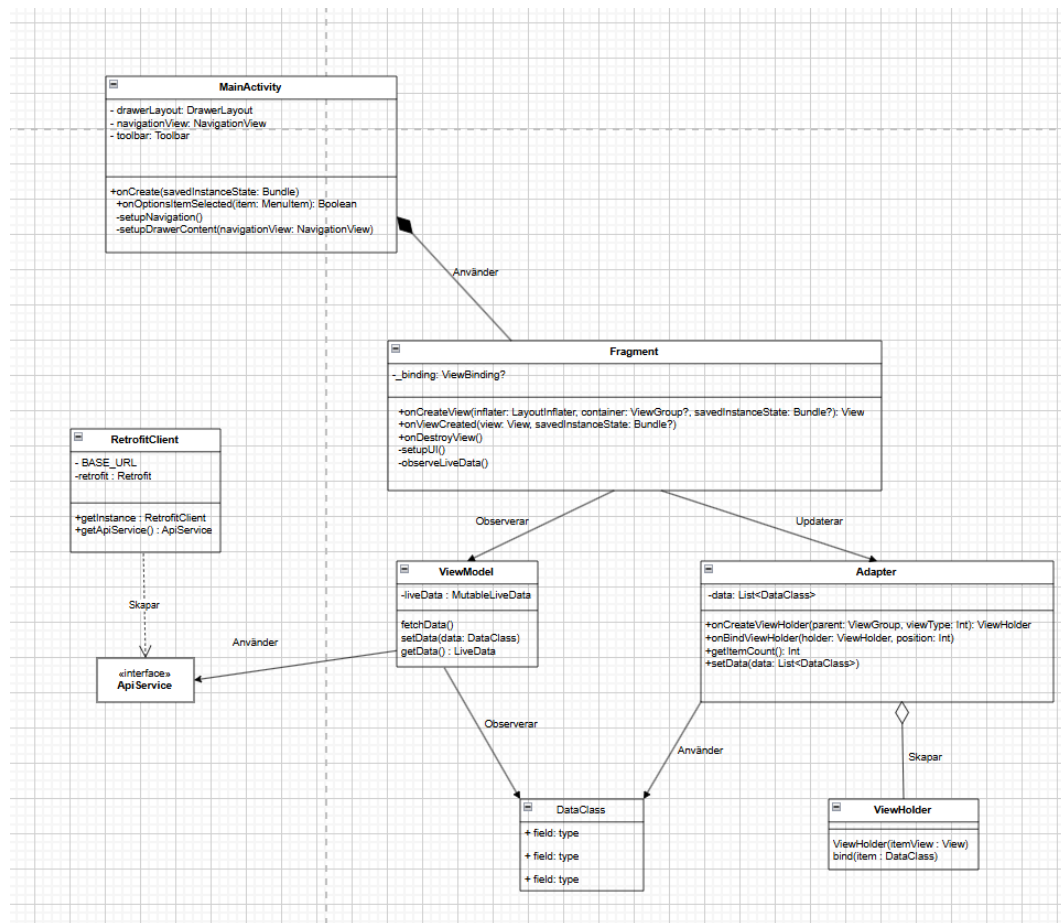
Här raderas förra användarnamnet och ditt nya blir aktivt. I detta fall ändrades namnet till *linuss* och sedan skickades ett meddelande i samma chatt som tidigare vilket gav följande resultat (se figur 16).



Figur 16 - bilden visar det tidigare chattrummet, där det nu har ett nytt användarnamn.

### 3.3 UML-diagram

För att förstå hur vår kod hänger ihop har det genomförts ett övergripande UML-diagram (se figur 17). Diagrammet visar de centrala komponenternas roller samt hur de är kopplade till varandra.

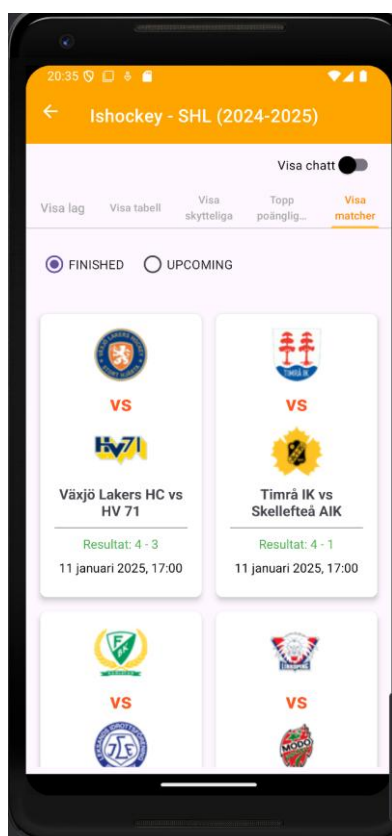


Figur 17 - bilden visar ett övergripande UML-diagram över applikationen

UML-diagrammet visar inte XML-filer då de endast innehåller Grafisk design och inte har något med logiken att göra. Det som inte fångas i UML-diagrammet är vårt användande av shared preferences, recyclerView och vår databas. Men i övrigt begriper diagrammet själva designvalet.

## 4 Analys och diskussionen

I början av arbetet hade vi en tydlig plan om vad vi vela göra, men det första problemet vi stötte på för att representera data ifrån ett API. Detta löstes sedan och efter man lärt sig representera data så blev det lättare när vi skulle visa mer nya data. När det blev klart så lärde vi oss om recyclerview, vilket blev en fördel då vi lätt kunde representera data snyggare. Utöver de ville vi ha nå ytligare funktion, vilket blev ett chattrum för alla ligorna och alla meddelanden sparas strukturerat i en databas. Vi sparar även användarnamn som matats in med hjälp av shared preferences. Utöver detta har vi även tagit hänsyn till språk, hårdkodade strängar och objektorientering. Dessa saker är arbetets styrkor. Utöver detta så finns det såklart nackdelar. Exempelvis så dök det upp oförutsägbara problem. Ett exempel är att det inte går att översätta dessa radiobuttons och att de konstant står på engelska (se figur 18), finns även att flera användare kan ha samma användarnamn eftersom shared preferences sparar namnet lokalt på enheten som använder appen. Vi har också använt oss av "android:configChanges" som medför till applikationen inte följer android livscykeln.



Figur 18 – bilden visar applikationen på svenska men inte hos de radio buttons.