



KATTEGATTGYMNASIET

Firestore

Richard Johansson

Läsåret 20-21

Firestore

En analys mellan olika serverspråk

Handledare: Fredrik Persson

ABSTRACT

Webservers are the foundation of our internet. For example, without webservers we wouldn't have any of our big social medias, like YouTube or Facebook. To send, receive and store data, to and from different users, it's what it all is about.

This report will compare two different ways to accomplish a working modern website, which will include data management, data storage and security rules such as authorization and authentication. This project will feature the server backend language; PHP, and the backend as a service; Firebase which will be based on the language JavaScript.

The report will explore the differences and similarities between Firebase and PHP as well as pros and cons; where the main points are the ease of starting a Firebase project and the flexibility as well as control you have over a PHP-project.

Keywords: PHP, Firebase, Webserver, Backend as a Service

INNEHÅLLSFÖRTECKNING

1	INLEDNING.....	- 1 -
1.1	BAKGRUND.....	- 1 -
1.1.0	Notering.....	- 1 -
1.1.1	Firebase.....	- 1 -
1.1.2	PHP.....	- 2 -
1.2	SYFTE.....	- 2 -
1.3	FRÅGESTÄLLNINGAR.....	- 2 -
1.4	AVGRÄNSNINGAR.....	- 2 -
2	METOD.....	- 4 -
2.1	VAL AV ARBETE.....	- 4 -
2.2	INLÄRNING AV FIREBASE & PHP.....	- 4 -
2.2.1	Vad behöver man lära sig?.....	- 4 -
2.2.2	Hur lärde jag mig?.....	- 4 -
2.3	WEBBSIDA & ANALYS.....	- 4 -
2.4	VERTYG.....	- 5 -
2.5	SCHEMA.....	- 5 -
3	RESULTATREDOVISNING.....	- 7 -
3.1	UPPSTART AV PROJEKT.....	- 7 -
3.1.1	Firebase.....	- 7 -
3.1.2	PHP.....	- 9 -
3.1.3	Starta sin webbsida.....	- 9 -
3.2	HUVUDEL.....	- 10 -
3.2.1	Autentisering.....	- 10 -
3.2.1.1	REGISTRERING.....	- 11 -
3.2.1.2	INLOGGNINGSPROCESS.....	- 14 -
3.2.1.3	ANVÄNDARINFORMATION.....	- 15 -
3.2.2	CRUD – Create, Read, Update, Delete.....	- 16 -
3.2.2.1	CREATE.....	- 17 -
3.2.2.2	READ.....	- 20 -
3.2.2.3	UPDATE.....	- 21 -
3.2.2.4	DELETE.....	- 23 -
3.2.3	Övrigt.....	- 24 -
3.2.3.1	ASYNC-AWAIT.....	- 24 -
3.2.3.2	FETCH.....	- 24 -
3.2.3.3	TEMPLATE.....	- 25 -
3.3	SÄKERHET.....	- 25 -
3.3.1	Säkerhetsanordningar.....	- 25 -
3.3.1.1	AUTENTISERING.....	- 25 -
3.3.1.2	KLIENT-SÄKERHET.....	- 26 -
3.3.1.3	AUKTORISERING.....	- 27 -
3.3.2	Felhantering.....	- 28 -
3.3.3	IP/Domän-bindning för Firebase.....	- 29 -
4	DISKUSSION OCH SLUTSATS.....	- 30 -
5	KÄLL- OCH LITTERATURFÖRTECKNING.....	- 31 -
6	BILAGOR.....	- 32 -

6.1	FIREBASE - JAVASCRIPT.....	- 32 -
6.2	FIREBASE – HTML, HTML-MALLAR & CSS.....	- 46 -
6.3	KLIENTVY.....	- 56 -
6.3.1	<i>Startsida</i>	- 56 -
6.3.2	<i>Bok-vy</i>	- 56 -
6.3.3	<i>Login-vy</i>	- 57 -
6.3.4	<i>Registrerings-vy</i>	- 57 -
6.3.5	<i>'Add a username'-vy</i>	- 58 -
6.3.6	<i>Lägg till bok i bibliotek-vy</i>	- 58 -

1 INLEDNING

1.1 BAKGRUND

Webbserverar är grunden för dagens hemsidor. Oavsett om det är Facebook, Youtube eller recept.se, så använder alla olika typer av webbserverprogrammering. Utmatning av data i nyhetsflöden, lagring av data, användarkonton och mycket mer sköts med olika typer av webbserverar som är programmerade på olika sätt för att uppnå olika mål.

Ett av de nyare sätten att göra detta är med Firebase (se rubrik 1.1.1) som kommer att jämföras mot den mer traditionella webbserverprogrammeringen i språket PHP (se rubrik 1.1.2). Språket som denna rapport kommer huvudsakligen grunda sig mot är Firebase, det kommer ske mer förklaringar i helhet över hur Firebase-hemsidan fungerar än hur den motsvarande PHP-sidan fungerar, med denna rapport är ingen guide till hur man skriver Firebase.

Mitt namn är Richard Johansson, jag avslutar för närvarande mitt tredje och sista år på Kattegattgymnasiet inom teknikprogrammet med inriktning Information och Kommunikations teknik. Jag har valt att fördjupa mig inom webbserverprogrammering då jag personligen tycker det är intressant, samt för att jag vill vidja mina vyer inom webbserverns värld, men även för att kunna anpassa mina handlingsätt för att maximera min effektivitet. Varför jag valde att lära mig Firebase är av rekommendationer samt att det är ett ypperligt tillfälle att utveckla mina javascript kunskaper.

1.1.1 Notering

I denna rapport kommer jag att referera till Firebase (baserad på JavaScript) och PHP som två olika programmeringsspråk även om det inte är av helt korrekt användning. Firebase per sig är inget språk, utan kan appliceras till olika programmeringsspråk (inklusive till PHP), och när det refereras som ett språk i detta projekt är det mer yppat till att JavaScript som detta Firebaseprojekt kommer att appliceras till, är ett separat programmeringsspråk, skilt från PHP.

Rapporten är även skriven utifrån mina egna upplevelser, svårigheter och utmaningar. Därav av Resultatredovisning baserad på mina egna arbeten med mig själv som primärkälla om inget annat har angetts.

1.1.2 Firebase

Enligt Doug Stevenson (2018, 25 sep) i sin artikel *What is Firebase? The complete story, abridged.* hos *medium.com* drar han liknelsen till att Firebase är en verktygslåda, en verktygslåda som innehåller verktyg som är gjorda för webbserverprogrammerare. Eftersom i vanliga fall tvingas programmerarna att bygga dessa verktyg själva, inte för att de vill bygga dem utan för att de måste göra det, för att ha en funktionell hemsida.

Det är det geniala med Firebase, man får en färdig verktygslåda som bara är att använda. Funktioner som Firebase ger är bland annat analysverktyg, användaridentifiering, databaser, fillagring, push notiser och mycket mer.

Att använda fördefinierade funktioner har sina för och nackdelar, vilket jag kommer att utforska i denna rapport.

Från Firebase hemsida (firebase.google.com, "2021") kan man se att Firebase kan appliceras till ett flertal programmeringsspråk, såsom JavaScript, Swift, Java, Python, PHP med mera. Det går att applicera i princip alla miljöer.'

Firestore är i mindre skalor gratis att använda. Däremot om man bygger en hemsida som kommer ha mindre till måttlig användning, kommer gratisplanen vara enkel att överstiga.

1.1.3 PHP

PHP stod från början för *Personal Home Page*, men har senare byts till *Hypertext Preprocessor*. PHP, som är ett skriftspråk används främst för webbservrar där dess uppgift är att genomföra hela eller delar av ett HTTP-respons. PHP som webserver motsvarar ungefär 80% av serverkodade webbplatser enligt w3techs (w3techs.com, 2021). Hos Wikipedia ("PHP", 2021, 1 april) står det att PHP inte är begränsad endast till webbservrar utan kan även användas till programmeringsuppgifter exempelvis som fristående grafikapplikation eller robotisk drönarkontroll.

PHP är en helt gratis, open-source general purpose skript språk.

1.2 SYFTE

Syftet med detta gymnastikprojekt är skapa en djupare förståelse för hur olika typer av webbservrar fungerar samt olika sätt att programmera dem på. En stor del av detta projekt kommer att gå ut på att lära sig Firebase i kombination med JavaScript, som Firebase i mitt fall kommer vara grundat på.

Efter att grunden har etablerats kommer det ske en jämförelse mellan Firebase och det traditionella PHP för att analysera ett flertal frågor som kan ses under rubrik **1.3 Frågeställningar**. Under och efter analysen av frågeställningarna kommer det ske en slutsats angående språkens för och nackdelar inom olika områden.

1.3 FRÅGESTÄLLNINGAR

- För och nackdelar med att använda Firebase i stället för PHP.
- Är det värt att lära sig Firebase.
- Olika situationer där ena eller andra språket är fördelaktig.
- I framtida bruk, vilket språk kommer jag helst att använda.

1.4 AVGRÄNSNINGAR

Det kommer ske en analys mellan Firebase och PHP, men Firebase är huvudsyftet med rapporten och kommer därav ha mest fokus och förklaringar.

Även om det sker genomgåendes förklaring för olika funktioner så är denna rapport ingen guide för att börja med vare sig PHP eller Firebase.

2 METOD

2.1 VAL AV ARBETE

I början av höstterminen var jag osäker på val av gymnasiearbete. Jag var osäker på vad jag ville göra och exakt vad som krävdes. Men efter en del diskuterande och många tankegångar fick jag rekommendationen att arbeta med Firebase. I början av detta projekt hade jag endast minimal förståelse för hur serverprogrammering fungerade och då fick jag en förklaring för vad Firebase var som lät intressant.

2.2 INLÄRNING AV FIREBASE & PHP

2.2.1 Vad behöver man lära sig?

För att göra en analys mellan Firebase och PHP behövde jag lära med ett par olika saker.

Dels behövde jag lära mig hur Firebase fungerar, eftersom det är vad projektet handlar om. Jag behövde lära mig JavaScript, språket som jag har använt för att koda med Firebase. Sist behövde jag lära mig PHP, eftersom det är vad projektet kommer att jämföras emot.

2.2.2 Hur lärde jag mig?

Det finns ett flertal källor man kan använda för att lära sig ovanstående punkter.

Tas listan i omvänd ordning, så använde jag mig utav tre sätt för att lära mig PHP. Min huvudsakliga inlärningskälla är våra lektioner inom webbserverprogrammering, vilket hölls av min gymnasieprojektshandledare Fredrik Persson. De övriga sätten där jag lärt mig PHP är delvis Youtube som har många genomgångar man kan följa, och delvis hemsidan StackOverflow som är till oerhörd hjälp inom alla typer av programmering.

JavaScript som mitt Firebase är grundat på, lärde jag mig dels av min handledare Fredrik Persson i samband med Firebase. Men jag har lärt mig mycket JavaScript från Youtube-kurser, StackOverFlow men också min far som är väldigt kunnig inom just JavaScript.

Sist men inte minst har jag lärt mig Firebase inför detta projekt. För att komma igång fick jag privatlektioner av min handledare F. Persson, det gav mig grunderna för att påbörja en hemsida baserad på Firebase. Sedan precis som ovanstående punkter så har Youtube varit till stor hjälp i samband med StackOverFlow. Utöver de så har firebase.google.com, Firebase egna hemsida används dels för att starta projektet, men dels för att använda projektet i sig, för att inte nämna alla funktionsgenomgångar som Firebase har.

2.3 WEBBSIDA & ANALYS

För att göra en jämförelse mellan Firebase och PHP valde jag att göra en hemsida med ett par krav. Kraven för hemsidan är följande:

- **CRUD** – Create, Read, Update, Delete. Standard för en modern hemsida. Det innebär att man ska kunna läsa in hemsida, vanligtvis med en typ av template. Man ska kunna skapa information till hemsidan inkluderande lagring av skapad information, uppdatera informationen samt ha möjligheten att radera tidigare skapad information.
- **Autentisering och auktorisering.** Registrering och hantering av användarkonton som ger vissa förmåner, exempelvis att kunna skapa och uppdatera filer.
- **Säkerhet.** Hemsidorna ska uppfylla standard säkerhetskrav

Genom att följa dessa krav har man grunden för vilken social webbplattform som helst, oavsett om det är Youtube eller Facebook. Jag valde att skapa en sida där man kan lägga in böcker, och bokserier. Under mina lektioner om PHP gjorde jag även en sådan hemsida som slutprojekt, så det var ett klart val att göra samma projekt fast i Firebase. Detta medgav att delar av de visuella element samt strukturen är något som jag redan hade planerat vilket sparade mig arbete.

Firebase-hemsidan började jag med efter jullovet, under första halvan av vårterminen. Medan PHP hemsidan var färdig i slutet av höstterminen.

2.4 VERTYG

Verktygen som användes under detta projekt är Visual Studio Code. [Firebase.google.com](https://firebase.google.com) – Firebase hemsida vilket ger åtkomst till att skapa projekt, redigera databasregler med mer. Samt [Repl.it](https://repl.it) även kallad Replit. Det är en onlinetjänst där jag skrev mitt PHP projekt.

2.5 SCHEMA

Jag gjorde ett kvartalschema.

- **Årskurskvartal 1:**

Förstå vad webbserverprogrammering innebär. Vad är PHP, vad är Firebase. Samt att börja lära sig Firebase i kombination med JavaScript.

- **Årskurskvartal 2**

Lära mig PHP och Firebase. PHP-projektet färdiggjordes under detta kvartal.

- **Årskurskvartal 3**

Skriva Firebase hemsidan samt rapport.

- **Början av kvartal 4**

Skriv färdigt och lämna i rapport.

Övrigt:

Om det är något som jag i efterhand hade ändrar skulle det vara Rapportskrivningen. Den börjades lite innan kvartal 4, men jag trodde att

det skulle vara mer tid fram tills inlämning, men det fanns tillräckligt med luft i schemat för att ta hand om denna förändring.

3 RESULTATREDOVISNING

Under resultatredovisning kommer de intressanta delarna av koden bakom hemsidorna att redovisas. Under redovisningen kommer kodblockens funktioner att förklaras och jämföras, det kommer även förekomma korta slutsatser per jämförelse. För helhetsbedömning, slutsats och direkta svar för frågeställningarna, se nästa rubrik; **4 Diskussion och Slutsats**.

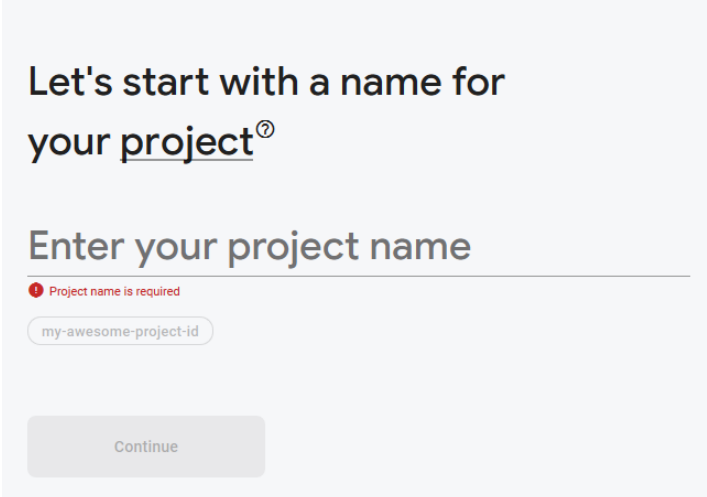
Se hela Firebase koden och hemsidan under punkt **6.1** och **6.2**.

3.1 UPPSTART AV PROJEKT

3.1.1 Firebase

Att starta ett Firebase-projekt är ganska enkelt och helt gratis. För att starta behöver man logga in på firebase.google.com och trycka på knappen 'create a project'. Därefter presenteras en relativt enkel startprocess, man anger projektnamn (Bilaga 7.1), därefter har man ett projekt. För att göra projektet användbart med JavaScript behöver man skapa en webbapplikation (Bilaga 7.2), vilket görs genom att trycka på webbkods-ikonen på projektets startsida. Man får upp nya alternativ där man ger webbapplikationen ett namn för att därefter få ut en configurationsnyckel (Bilaga 7.3) som man klistrar in i sin webbsida. Konfigurationsnycklen är vad som ger en hemsida, oberoende av var servern körs, tillgång till att läsa, skapa och ändra filer hos ett Firebaseprojekt. Därav, när ens hemsida är färdig anger man ens server IP och säger att den IP är den enda som får röra en Firebaseprojekt, detta kommer att redovisas i punkt **3.3.1.3**.

Bilaga 7.1



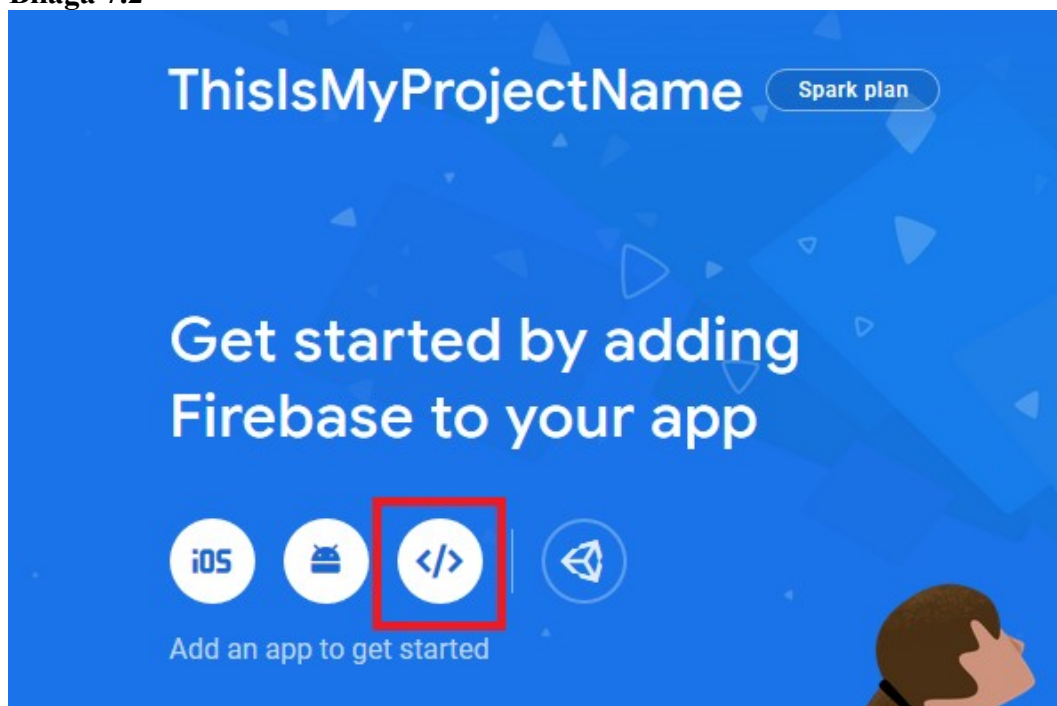
Let's start with a name for
your project®

Enter your project name


Project name is required

my-awesome-project-id

Continue



Bilaga 7.3

 **Add Firebase to your web app**

✓ Register app

2 Add Firebase SDK

Copy and paste these scripts into the bottom of your <body> tag, but before you use any Firebase services:

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/8.3.2/firebase-app.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
https://firebase.google.com/docs/web/setup#available-libraries -->

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyCWorJpTB__wdQOrY60Mh1unNTWpivfpsk",
    authDomain: "thisismyprojectname-bb2c5.firebaseio.com",
    projectId: "thisismyprojectname-bb2c5",
    storageBucket: "thisismyprojectname-bb2c5.appspot.com",
    messagingSenderId: "1053344866306",
    appId: "1:1053344866306:web:9aebb0721984b856e5071b"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```

Learn more about Firebase for web: [Get started](#), [Web SDK API reference](#), [Samples](#)

Continue to the console

OBS! Denna konfigurationsnyckel kommer från ett tillfälligt exempelprojekt och kommer inte kunna appliceras.

3.1.2 PHP

Att starta ett PHP projekt kan göras på ett par olika sätt. Dels kan det göras lokalt, dels kan det göras online. Väljer man att köra PHP lokalt kräver det installation av en Wamp-server, vilket är ett separat program. Kör man PHP online kan man göra det via flera olika tjänster, tjänsten som PHP-hemsidan använder är Replit; en gratis, webbsida stödjer webbservrar som PHP.

3.1.3 Starta sin webbsida

Firebase projektet skrivs i programmet Visual Studio Code, vilket är gratis att ladda ner och använda.

När man arbetar med webbplatser med Firebase grundar sig allt i ett html-formulär som hämtar alla JavaScript funktioner och kör koden. För att få tillgång till Firebase-funktionerna behöver man importera de

Firebasebibliotek (bilaga 7.4) som man ska använda till sitt projekt. Det ska ske innan man anger sin konfigurationsnyckel och innan man använder någon Firebasefunktion. Efter ens import av Firebasebiblioteken anges ens konfigurationsnyckel, det kan ske genom att klistra in en script-tag med nyckeln, alternativt importerar man ett dokument med nyckeln för snyggare och mer läsbar kod.

Bilaga 7.4

```
<!-- Gets firebase libraries -->
<script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-firestore.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-auth.js"></script>
```

Det kan tolkas som tidskrävande att starta en firebase webbsida. Men det är bara att kopiera och klistra in.

Att starta en PHP-webbplats via repl.it tar knappt någon tid. Man trycket på en knapp, namnger sitt projekt och sedan kan man skriva sin hemsida.

Efter alternativt under utvecklingen av sin hemsida kan man ladda upp sin hemsida till en egen domän. Det är efter detta steg som man kan binda IP adressen för vilka IP:s som får åtkomst till ens Firebaseprojekt. Denna funktion kommer att visas längre ner i punk **3.3.1.3**.

3.2 HUVUDEL

Efter att ens webbsida är uppsatt kan man börja med sin kod. HTML-sidan för Firebaseprojektet kan ses vid i punkten **6.1.2**.

De huvudsakligen intressanta punkterna för en webbsida, vilket kommer presenteras och jämföras är, utan inbördes ordning följande:

- Autentisering
- Registrering
- Inloggning
- CRUD – Create, Read, Update, Delete
- Säkerhet
- Templetting (Fetch / Include)
- Övriga funktioner

3.2.1 Autentisering

En generell fördels som anses ligga hos Firebase är dess enkelhet för autentisering, med andra ord; enkelheten att skapa användare, logga in användare, samt att kolla om en användare är inloggad. Autentisering är en av punkterna där Firebase skiner som starkast. Detta kan enkelt styrkas från flera källor exempelvis back4app's artikel "What is Firebase Authentication?" skriven av Jessica Clark eller Lucas Do's artikel "Implement Authentication in Flutter with Firebase Authentication" hos medium.com. Både dessa understryker enkelheten att dels komma igång med Firebase, men särskilt autentiseringsstadiet som i andra fall när man skriver den själv kan lätt bli krånglig.

3.2.1.1 REGISTRERING

Tittar man på registreringsprocessen i Firebase, så är den precis lika enkel som de ovanstående källorna hävdar.

Bilaga 7.5

```
const promise = firebase.auth().createUserWithEmailAndPassword(email, pwd);
```

På en enda rad, har man registrerat en användare med Firebase, detta är det som snart kommer att jämföras mot PHP:s sätt att genomföra detta. Man kallar på funktionen *createUserWithEmailAndPassword* som beskriver sig själv, hämtas från *auth()*-funktionsklassen, vilket i sin tur hämtas från *firebase* klassen, registreringsprocessen ger även automatiskt ett unikt id till användaren. I bilaga 7.5 tas en *promise* emot som sedan används för att hantera eventuella fel, exempelvis om användarens mejl är inkorrekt skriven eller om användaren inte angav ett lösenord. Allt sådant sköts automatiskt, och det är ett av argumenten för Firebase. Även om mycket sköts automatiskt så finns det fortfarande saker som behövs göras runtomkring registreringsprocessen, vilket kan ses i bilaga 7.6, det som görs är en kontroll om användaren redan är inloggad på rad 84, man kollar om *currentUser* i *auth()*-funktionsklassen har ett värde, saknas värde är användaren inte inloggad och då fortsätter funktionen. Sedan på rad 90 och framåt hämtas registreringsinformation för att sedan matas in på rad 94. Rad 95 och 96 tar hand om en callback-funktion med *promise* vilket beskrevs ovan. Bakom kulisserna tar Firebase och lägger till den nyregistrerade användaren i en lista som man kan komma åt från Firebases hemsida.

Bilaga 7.6

```
80 //registers and adds username
81 ✓ async function registerUserInFirebase(event) {
82 ✓   try {
83     event.preventDefault();
84 ✓     if (firebase.auth().currentUser == null) {}
85 ✓       console.log(
86         'Registering when not logged in',
87         firebase.auth().currentUser
88       ); // expected null
89       console.log(this);
90       const email = this.email.value;
91       const pwd = this.pwd.value;
92       const promise = firebase
93         .auth()
94         .createUserWithEmailAndPassword(email, pwd);
95       promise.catch((e) => alert(e.message));
96       if (!promise.catch()) removeElement('#registerId');
97     }
98 ✓   } else {
99     console.log('User cannot register when signed in!');
100   }
101 ✓ } catch (error) {
102   console.log(error);
103 }
104 }
```

Ska man registrera en användare i PHP krävs det lite mer kod, finnes och planering. Fullständiga PHP-registreringsprocessen finns under Bilaga 7.7 längre ner i rapporten. Precis som i Firebase kontrolleras det om man är inloggad, i PHP-projektet sker dit inför inladdningen av registreringsformuläret. För att specifikt registrera ett konto krävs tre saker. Det krävs ett id, en email och ett lösenord. Lösenordet och mejlen skickas med till funktionen, vilket kan jämföras med hämtningen av registreringsinformation ovan i Firebase-projektet. På rad 16 hämtas en lista med användare för att sedan kontrollera att den registrerande användaren inte redan har ett konto, vilket Firebase gör automatiskt. På rad 24 hashas lösenordet, vilket är en del av säkerhetsprocessen, man ska aldrig lagra lösenord i klartext och detta är ett sätt att undvika det, Firebase sköter även denna process åt en. Rad 43 innehåller en if-sats som kollar om användaren redan existerar, finns användaren redan skickas de tillbaka med ett felmeddelande. Vid rad 45 skapas ett unikt id till användaren, i detta exempel är id:t baserat på användarens mejl men man kan även basera id:n på klockan, detta steg gör Firebase också automatiskt. Forsätter man till rad 49 skapas en ny fil för användaren med dess information vilket omvandlas till ett jsonobjekt. Om något av stegen returnerar ett fel kommer användaren att skickas tillbaka till registreringsidan med ett felmeddelande.

Vilket är uppenbart om man tittar på bilagorna är att Firebases registrering sker inom en funktion på ca 20 rader, medans PHPs registreringsprocess tar upp ett helt A4-papper.

Bilaga 7.7

```
1 <?php
2
3 session_start();
4
5 if (count($_POST)){
6
7     require_once("userChecker.php");
8
9     $user = $_POST;
10    if($user['email'] == null || $user['username'] == null || $user['password'] ==
11    null){
12        header("Location:register.php?mes=regFail&res=incorrectInput");
13        exit();
14    }
15    $user['email'] = strtolower($user['email']);
16
17    foreach($userArr as $item){ //replaces filename with file
18        if($item == md5($user['email']).".json"){
19            header("Location:register.php?mes=regFail&res=userExist");
20            exit();
21        }
22    }
23
24    $user["password"] = password_hash($user['password'], PASSWORD_DEFAULT, ["cost"=>
25    14]);
26
27    // Get existing userList from file
28    $userArr = scandir("../db/users/");
29    $userArr = array_diff($userArr, array('..', '.'));
30
31    $LOU = [];
32    foreach($userArr as $item){ //replaces filename with file
33        $LOU[] = json_decode(file_get_contents("../db/users/".$item)); //list of users
34    }
35
36    //checks if username is already in use
37    if($name = doesUsernameExist($user, $LOU)){
38        header("Location:register.php?mes=regFail&res=unEx");
39        exit();
40    }
41
42    // Check if email is already registered
43    if (!$email = doesUserExist($user, $LOU)) {
44
45        $user["uid"] = md5($user["email"]); // creates a unique id based on mail
46
47        $userFile = $user;
48
49        file_put_contents("../db/users/".$user["uid"].".json", json_encode($userFile,
50    JSON_PRETTY_PRINT));
51
52        header("Location:login.php?mes=regSucc"); //redirects you to the main site
53        exit();
54    }
55    else {
56        header('Location:register.php?mes=regFail&res=uEx');
57        exit();
58    }
59 }else{
60     header('Location:register.php?mes=regFail'); // if no post, sends away
61     exit();
62 }
63 }
64
65 ?>
```

3.2.1.2 INLOGGNINGSPROCESS

Att logga in en redan registrerad användare i Firebase är ännu enklare än att registrera användaren vilket kan ses i bilaga 7.8. Man behöver ta in inloggningsinformationen, vilket hämtas från ett formulär oavsett skriftspråk. När man har informationen anropar man funktionen i bilaga 7.9. På en rad, bortsett från införsel av inloggningsinformationen, kan man logga in en tidigare registrerad användare.

Bilaga 7.8

```
62 // Log in the user
63 function loginUser(event) {
64   try {
65     event.preventDefault();
66     console.log(this);
67
68     const email = this.email.value;
69     const pwd = this.pwd.value;
70
71     const promise = firebase.auth().signInWithEmailAndPassword(email, pwd);
72     promise.catch((e) => alert(e.message));
73
74     removeElement('#loginId');
75   } catch (error) {
76     console.log(error);
77   }
78 }
```

Bilaga 7.9

```
const promise = firebase.auth().signInWithEmailAndPassword(email, pwd);
```

Firebase sköter automatiskt allt runt om inloggningen, senare under Resultatredovisnings-rubriken kommer det komma autentiseringslyssnare som körs då man loggas in eller ut. Kollar man på rad 72 i bilaga 7.8 sköts felhantering vilket skickas till användaren, exempelvis om man har fel lösenord, om användaren redan är inloggad eller inte existerar, väldigt smidigt.

Att logga in en registrerad användare är i PHP betydligt smidigare än att registrera användarna, vilket kan ses i bilaga 7.10. Rad 7 till 12 kontrollerar att användaren har skrivit in nödvändig information. På rad 19 granskas det om användaren existerar, sedan på rad 22 används PHP-funktionen *password_verify* där lösenordet som användaren angett verifieras mot det lagrade hashade lösenordet vilket angavs i registreringsprocessen. Vid rad 24 avstäms om lösenorden matchade, om det gjorde det ska användaren loggas in vilket sker på rad 26, man sätter en sessionsvariabel till sann. Därefter läggs lite övrig information med i sessionsvariabeln vilken används under andra processer.

Inloggningen per sig är enkel i PHP, men det är allting runtom som är besvärligt. Man måste verifiera inmatningsvärden och sköta felmeddelanden, vilket Firebase sköter helt och hållet åt programmeraren.

Bilaga 7.10

```

1 <?php
2
3 session_start();
4
5 require_once('userChecker.php');
6
7 if($_POST){
8     if (isset($_POST['email']))
9     && isset($_POST['password'])
10    && ($_POST['email']!=null)
11    && ($_POST['password']!=null)
12    )
13    {
14        $tryLogin = $_POST;
15        $tryLogin['email'] = strtolower($tryLogin['email']);
16        $uid = md5($_POST['email']);
17        $tryUser = json_decode(file_get_contents('../db/users/'.$uid.'.json'));
18
19        if($currentUser = doesUserExist( $tryLogin, $tryUser)){//!throws me out
20
21            //when we've found a user
22            $pwdCheck = password_verify($tryLogin['password'], $currentUser['password']);
23
24            if($pwdCheck){
25                session_start();
26                $_SESSION['loggedIn'] = true;
27                $_SESSION['username'] = $currentUser['username'];
28                $_SESSION['uid'] = $currentUser['uid'];
29                header('Location:../?mes=loggedIn');
30                exit();
31            }else{
32                header('Location:login.php?mes=loginFail'); // if no post, sends away
33                exit();
34            }
35        }else{
36            header('Location:login.php?mes=loginFail');
37            exit();
38        }
39    }else{
40        header('Location:login.php?mes=loginFail');
41        exit();
42    }
43 }else{
44     header('Location:login.php?');
45     exit();
46 }
47 ?>

```

3.2.1.3 ANVÄNDARINFORMATION

Extra användarinformation, det kan exempelvis vara en roll som användaren får såsom administratör eller standard-användare, men det kan även innebära saker som användarnamn. Extra användarinformation är något som har använts i både det refererade Firebaseprojektet, och i PHP-projektet. Att tillägga extra information till användarfilen, vilket i PHP-projektet görs under registrering. I bilaga 7.10 längre ner, vilket visar hela redigeringsprocessen läggs i detta fall användarnamn till på rad 9, när man hämtar sin information som tidigare kommer från ett formulär. Det innebär även att, vill man tillföra fler informationsfält är det bara att lägga till fler fält i registreringsformuläret, det är enkelt skalbart. Utöver det, med servertillgång som programmerare, kan man även i efterhand öppna en användares dokument och ändra samt lägga till information, exempelvis lägga till satsen *'roll = admin'*. Sedan när användaren ska göra en uppgift kan servern verifiera om användaren är en administratör och därefter jaka eller neka åtkomst.

Det är vid extra användarinformationstillägg som Firebases baksida visas. Man får många fördelar i enkelhet av kodskrivning, på bekostnad av flexibilitet och frihet.

Hur användare förvaras i PHP-projektet och i Firebase-projektet är sker på två olika sätt. PHP-projektet förvarar sina användare i json-filer, som programmeraren bakom projektet skapar i registreringsprocessen. Därav har programmeraren full kontroll över dessa dokument. I Firebase däremot, så lagras användarna i en lista, som programmeraren inte har tillgång att ändra information i. Vilket gör att man måste göra dubbla användarlistor vilket medför ökad datatrafik, mer lagringsutrymme och mer som kan gå fel.

Som förklarat ovan, vill man lägga till användarnamn som mer användarinformation i PHP är det bara att skicka med den användarinformation in i datalagringsmetoden för användarfilen. I Firebase är man begränsad till att endast skicka med mejl och lösenord (bilaga 7.5 längre upp). Skickar man in fler värden svarar registreringsfunktionen bara med ett felmeddelande. Utöver detta har man även problemet, att av säkerhetsskäl låter man inte en icke inloggad användare skapa eller skriva till filer. Detta i sin tur medför att man kan inte direkt skapa en separat användarfil direkt vid registreringen, då registreringen inte kan användas med `async-await` vilket förklaras i en annan punkt. Att man inte bara kan vänta på registreringsfunktionen tills att man är inloggad innebär att man måste kolla om användarens fil existerar vid varje inloggning. Därefter, om den existerar kan man hämta ut ett användarnamn, alternativt om den inte finns kan man göra en pop-up ruta som ber användaren att fylla i ett användarnamn, och i samband med användarnamnet skapar man en användarfil. Efter att användarfilen har skapats kan man som programmerare öppna filsystemet och lägga till taggen `'roll = admin'` precis som i PHP, men endast efter att användaren har loggat in för första gången efter registrering.

3.2.2 CRUD – Create, Read, Update, Delete

Förkortningen CRUD är en av grunderna för en oerhörd mängd hemsidor, oavsett om det är en social media, exempelvis Facebook, eller om det är en dagstidning såsom Svenska Dagbladet kan man skapa, läsa, uppdatera och ta bort information.

På Facebook kan alla registrerade användare skapa ett eget inlägg, på SvD kan journalister skriva och ladda upp sina artiklar; där har vi första bokstaven i CRUD; Create – skapa. R:et står för Read, att läsa eller hämta information. Om man följer exemplen ovan kan det vara att man läser en nyhetsartikel hos SvD eller ett inlägg på Facebook. Tredje bokstaven i CRUD står för Update – att uppdatera eller att ändra information; vilket kan motsvara att man redigerar ett av sina egna inlägg på Facebook eller uppdaterar en artikel som innehåller missinformation på dagstidningen. Sista bokstaven i CRUD står för Delete, att ta bort information. Det kan om man fortsätter med ovan exemplen vara att man raderar gamla ickerelevanta artiklar från en hemsida eller att man tar ner ett Facebook-inlägg som man tidigare har publicerat.

3.2.2.1 CREATE

I Bilaga 7.11 finns skaparfunktionen för Firebase-projektet. Den är väldigt simpel att följa. Funktionen är en `async-await`-funktion som kommer att förklaras noggrannare i punkt 3.2.3.1, men det som är bra att veta är att `async-await`-funktioner används för att kunna vänta på ett svar utan att stoppa upp kod-processen.

Funktionen i Bilaga 7.11 fungerar genom att ett formulär med värden kallar på `addBookToDataBase`-funktionen som på rad 9 till 19 namnger och lägger in värdena i ett objekt, rad 17 bland dessa rader lägger till skaparens användar-id. Sedan på rad 21 skickar man värdena till sin 'collection', vilket samlingarna i Firebase kallas för. `BookCol` på rad 19 är en variabel som har definierats tidigare i koden. Skulle man skriva raden utan förkortning hade rad 19 sett ut som Bilaga 7.12. `firebase.firestore()` anropar databasen, `collection('books')` innebär att det är bok-kollektionen som vi vill använda. Slutligen `.add(book)`, det engelska order för att lägga till där vi lägger till objektet `book`. Smidigt och enkelt.

Bilaga 7.11 – Create funktion Firebase

```
3 // Adds book to database,
4 async function addBookToDataBase(event) {
5   try {
6     event.preventDefault();
7     // Booktitle is required
8     if (this.addTitle.value !== null) {
9       let book = {
10         title: this.addTitle.value,
11         serie: this.addSerie.value,
12         searchSerieQuery: this.addSerie.value.toLowerCase(),
13         author: this.addAuthor.value,
14         year: this.addYear.value,
15         synopsis: this.addSynopsis.value,
16         timestamp: Date.now(),
17         uid: firebase.auth().currentUser.uid,
18         editors: {},
19       };
20       // with uid - relation, to know who the book belongs to
21       await bookCol.add(book);
22       removeElement('#bookAdderId');
23     }
24   } catch (error) {
25     console.log(error);
26   }
27 }
```

Bilaga 7.12

```
await firebase.firestore().collection('books').add(book);
```

När det kommer till standard PHP kommer Firebase ha ett stort övertag när det handlar om datalagring. Det är eftersom Firebase använder en databas, som i detta fall heter Firestore. Det finns databaser att använda till PHP-applikationer, men i PHP-projektet som Firebase jämförs med används det inte, men även om man hade använt en databas för PHP hade man behövt gör delar runtomkring sparandet som Firebase sköter automatiskt.

När det kommer till att lagra information som en användare anger måste man alltid tänkta på säkerhet och förhindra att en användare matar in och sparar farlig kod via kommande funktioner, säkerhet kommer att analyseras i punkt 3. 3, därav nämns det bara kort under kommande funktioner. Tittar man i Bilaga 7.13 ser man PHP-projektets spararfunktion som kräver en array med bok-information. Datan kommer att saneras, vilket sker genom att ta bort och ersätta tecken som krävs för att skriva skadliga funktioner på rad 12. Rad 13 och 14 sanerar *synopsis* (synopsis innebär kort beskrivning av en historia) separat från resterande data då man i detta fall vill ställa andra krav på den eventuella längre texten. På rad 20 läggs skaparens id i boken-informationen. Vid rad 21 och 22 skapas två arrays för framtida bruk vilket kommer att tas upp senare. I PHP-projektet lagras datan baserat på serier, därav kontrolleras det om en seriekatalog redan existerar med en början på rad 26, därefter skapas bokens separata katalog och eventuell seriekatalog om det behövs. Det sker även en kontrollering angående om boken redan existerar i biblioteket, detta sker fram till rad 43. På rad 45 omvandlas data-arrayn till ett JSON-object för att sedan skickas till sparnings funktionen som kan ses i bilaga 7.14, som endast sparar en objekt på angiven plats i biblioteket.

Att sanera datan som ska sparas, att sköta sparningsprocessen och ordna hur det ska skötas är en av de många fördelarna med Firebase. Som tidigare nämnt, delar av denna sparnings-processen skulle likna Firebases sparfunktion om man hade använt en databas, beroende på vilken databas och vad man har för inställningar sköts säkerheten också.

Bilaga 7.13 Create funktion PHP

```
9  function createBook($data) { //creates book and creates maptree for said book
10
11     $dataSyn = $data['synopsis'];
12     $data = str_repAQ($data); //sanitizes data with exception for synopsis
13     $data['synopsis'] = htmlentities($dataSyn); //encodes special chars for synopsis
14     $data['synopsis'] = nl2br(filter_var($data['synopsis'], FILTER_SANITIZE_STRING));
15
16     $title = strtolower($data['title']);
17     $serie = strtolower($data['serie']);
18
19     //adds extra data to bookfile
20     $data['create'] = $_SESSION['uid']; //creator of page
21     $data['edit'] = []; //editors of page
22     $data['reviews'] = []; // future array for reviews
23
24     //creates directory pathway if needed
25     $dir = "willBeFilled";
26     if ($serie != null) { //if dir doesn't exist and a series is set, create a new directory
27         if (!file_exists("../db/books/" . $serie . "/" . $title)) {
28             $dir = "../db/books/" . $serie . "/" . $title;
29             mkdir($dir, 0777, true);
30
31         } else { //else; book already exist, sends back because it should not be created again
32             header("location:create/createbook.php?error=bookExist");
33             exit();
34         }
35     } else if ($serie == null) { //if dir doesn't exist and series isn't set, create a new directory
36         if (!file_exists("../db/books/" . $title)) {
37             $dir = "../db/books/" . $title;
38             mkdir($dir, 0777, true);
39
40         } else { //else; book already exist, sends back because it should not be created again
41             header("location:create/createbook.php?error=bookExist");
42             exit();
43         }
44     }
45     $data = json_encode($data, JSON_PRETTY_PRINT); // encodes data
46     saveData($dir . "/", "book.json", $data); // saves data in potential new directory
47 }
```

Bilaga 7.14

```
49  function saveData($dir, $filename, $file) { // saves a $file to the $dir:ectory with a $filename
50      file_put_contents($dir . $filename, $file);
51  }
```

Bilaga 7.15

```
1  <?php
2
3  session_start();
4
5  if($_SESSION['loggedIn'] === true && $_POST){
6
7      require_once("../crud.php");
8
9      $data = $_POST;
10
11     createBook($data);
12
13     $data = str_repAQ($data);
14
15     $title = "title=".strtolower($data['title']);
16
17     if($data['serie']!=null){
18         $serie = "&serie=".strtolower($data['serie']);
19     }else{
20         $serie = "";
21     }
22
23     if($title){
24         header('Location:../../bookreader/?'.$title.$serie);
25         exit();
26     }
27     }else{
28         header("Location:../../login=signin");
29         exit();
30     }
```

3.2.2.2 READ

Datainläsning i Firebase kan skötas på ett par olika sätt. Man kan använda en *get*-metod, alternativt en *onSnapshot*-metod för att hämta datan. Skillnaden är att en *get*-metod endast hämtar datan medan en *onSnapshot*-metod lyssnar sedan efter ändringar vilket kommer att genomföras i punkt 3.2.2.3. Koden i Bilaga 7.16 hämtar ut och sorterar datan från en kollektion som är definierad i *bookLister*, datan placeras därefter in i en objekt-array. Slutligen skickas objekt-arrayn vidare till en mall och renderingsfunktion som ligger utanför bilagan. Nuvarande bilaga hämtar ut all data, ska man hämta ut data från ett specifikt dokument lägger man till *.doc('document-namn')* vid *bookLister* på översta raden.

Bilaga 7.16 Read – Firebase

```
booklistner.orderBy('timestamp', 'desc').onSnapshot((querySnapshot) => {  
  var books = [];  
  querySnapshot.forEach((doc) => {  
    books.push({ id: doc.id, data: doc.data() });  
  });  
});
```

Bilaga 7.17 Read – PHP

```
$bookIndex = [];  
if ($series = scandir("db/books/")) { //Gets directory to all available books  
  $series = array_diff($series, array('.', '.')); //delets the dots in start of scandirs, representing upper and current directory  
  foreach ($series as $serie) {  
    if (!file_exists("db/books/" . $serie . "/book.json")) {  
      $books = scandir("db/books/" . $serie);  
      $books = array_diff($books, array('.', '.'));  
      foreach ($books as $book) {  
        if (file_exists("db/books/" . $serie . "/" . $book . "/book.json")) {  
          $bookIndex[] = "db/books/" . $serie . "/" . $book . "/book.json";  
        }  
      }  
    } else {  
      $bookIndex[] = "db/books/" . $serie . "/book.json"; // puts directories of the book in an array  
    }  
  }  
}
```

För att hämta ut alla böcker ur PHP-projektet tas fil-biblioteket och skannas, på alla nivåer som eventuellt innehåller bok-filer. Böckerna läggs in som objekt i en array för att sedan användas i en mall och renderingsfunktion. För att läsa in en enskild fil i PHP krävs att man vet om det finns en serie, samt titeln på boken. Därefter får man ett JSON-objekt som man kan plocka ur informationen ifrån.

Bilaga 7.18 ReadFile – PHP

```
$json = json_decode(file_get_contents("../db/books/" . $tserie . $title . "/book.json"));
```

Konceptet i Firebase och PHP-projektet är lika, skillnaden är bara att i PHP-projektet krävs lite mer datahantering för att använda informationen.

3.2.2.3 UPDATE

I Bilaga 7.19, på rad 30 och 31 anges bok-kollektionen som ska uppdateras. Rad 41 hämtar eventuellt tidigare redigerare av boken som uppdateras, där rad 50 lägger till den nuvarande redigerande användaren tillsammans med en tidsstämpel. Från rad 53 till och med 60 kontrolleras om det finns nya värden, därefter läggs det in i ett objekt som på rad 62 uppdaterar dokumentet. Firebase skriver inte över en bok med en helt ny fil, utan uppdaterar bara de värden som har förändrats och som har skickats med i uppdateringsfunktionen.

Bilaga 7.19

```
29 async function updateBooks(event) {
30   var db = firebase.firestore(); // grants access firestore-database
31   var bookCol = db.collection('books');
32
33   //eventet är händelsen = submit
34   console.log('firebase.auth().currentUser;', firebase.auth().currentUser.uid);
35   console.log('this', this);
36
37   try {
38     event.preventDefault();
39
40     // gets editors
41     let editorDoc = await firebase
42       .firestore()
43       .collection('books')
44       .doc(this.updId.value)
45       .get();
46
47     arrEditors = [];
48     arrEditors = editorDoc.data().editors;
49
50     arrEditors[firebase.auth().currentUser.uid] = Date.now();
51
52     var updateObj = {};
53     if (this.updTitle.value) updateObj['title'] = this.updTitle.value;
54     if (this.updSerie.value) updateObj['serie'] = this.updSerie.value;
55     if (this.updSerie.value)
56       updateObj['searchSerieQuery'] = this.updSerie.value.toLowerCase();
57     if (this.updAuthor.value) updateObj['author'] = this.updAuthor.value;
58     if (this.updYear.value) updateObj['year'] = this.updYear.value;
59     if (this.updSynopsis.value) updateObj['synopsis'] = this.updSynopsis.value;
60     updateObj['editors'] = arrEditors;
61
62     await bookCol.doc(this.updId.value).update(updateObj);
```

Som förklarat ovan tar *onSnapshot()*-funktioner och lyssnar efter förändringar. I Bilaga 7.20 används också en *onSnapshot*-funktion, men den returnerar bara ändrade dokument, se rad 20. På rad 23 den uppdaterade boken in i en funktion som byter ut en tidigare renderad bok. I Firebase sker uppdateringar i realtid och sidan behöver därav inte laddas om.

Bilaga 7.20

```
17 bookLister.onSnapshot((docs) => {
18   var books = [];
19   // docChanges() gets changing documents, snapshot gets changed collection
20   docs.docChanges().forEach((element) => {
21     books.push({ id: element.doc.id, data: element.doc.data() });
22     console.log(books);
23     replaceClientBook(books);
24   });
25 });
```

Uppdateringsfunktionen som syns i Bilaga 7.21 i PHP är väldigt lik PHP:s skapar-funktion. Skillnaden är att uppdateringsfunktionen hämtar den data som användaren inte uppdaterar. Sedan sparar den en ny fil och skriver över den gamla bok-filen med samma spararfunktion som bokskaparen använder.

Bilaga 7.21 – PHP Update-funktion

```
53 function updateBook($data) { // updates book, note: name and series cannot be changed
54     $dataSyn = $data['synopsis'];
55     $data = str_repAQ($data); //sanitizes data with exception for synopsis
56     $data['synopsis'] = htmlentities($dataSyn); // synopsis are encoded instead of sanitized
57     $data['synopsis'] = nl2br(filter_var($data['synopsis'], FILTER_SANITIZE_STRING));
58     $title = strtolower($data['title']);
59     //checks if series is set (done automatically if book got a series)
60     if (strtolower($data['serie']) != "{{serie}}" && $data['serie'] != null) {
61         $serie = strtolower($data['serie']);
62     } else {
63         $serie = "";
64     }
65     if ($serie != null) { //checks if file is found and gets directory
66         if (file_exists("../db/books/" . $serie . "/" . $title)) {
67             $dir = "../db/books/" . $serie . "/" . $title;
68         }
69     } else { // If book got no serie, get directory;
70         $dir = "../db/books/" . $title;
71     }
72
73     //gets data and converts it into an array
74     $jsontdata = (array)json_decode(file_get_contents($dir . "/book.json"));
75
76     // replace data in json-file
77     foreach ($jsontdata as $key => $val) {
78         if ($val != $data[$key] && $data[$key] != null) {
79             $jsontdata[$key] = $data[$key];
80         }
81     };
82
83     //adds uid to editors if uid aren't already there
84     if (!in_array($_SESSION['uid'], $jsontdata['edit'])) {
85         $jsontdata['edit'][] = $_SESSION['uid'];
86     }
87     $json = json_encode($jsontdata, JSON_PRETTY_PRINT);
88
89     saveData($dir . "/", "book.json", $json); //overwrites old data
90 }
```

3.2.2.4 DELETE

Att ta bort ett dokument från Firebase görs, förutom enkel felhantering, på en enda rad. Man anger vilken kollektion och vilket id, som ska tas bort. I funktionen i bilagan nedan har id-variabeln fått namnet *docs*.

Bilaga 7.22

```
69 async function deleteDocs(docs) {
70     try {
71         await db.collection('books').doc(docs).delete();
72     } catch (error) {
73         console.log(error.message);
74     }
75 }
```

PHP-projektets borttagningsfunktion är relativt kort. Allt som görs är en kontroll om serien existerar, därefter startas en funktion *rrmdir* som kan ses i Bilaga 7.23 där de angivna biblioteksvägen skannas, går in i eventuellt hittade undermappar för att sedan tas bort. Anledningen till denna procedur är att PHP inte kan ta bort en mapp som innehåller föremål.

Bilaga 7.24

```
function deleteBook($title, $serie = null) { // deletes book from database
    $title = strtolower($title);
    if ($serie != null) {
        $serie = strtolower($serie);
    }
    $dir = "../db/books/";
    /*
    Checks if series got multiple books in them, if it.
    If true, serie-map should be kept.
    If count are not equal or less than 3 are because current( . ) and upper( .. ) directory are counted as well
    */
    if (count(scandir($dir . $serie)) <= 3 && $serie != null) {
        rmdir($dir . $serie);
    } else {
        rmdir($dir . $title);
    }
}
```

Bilaga 7.23

```
137 function rmdir($dir) { // removes directory-tree and everything inside it
138     if (is_dir($dir)) {
139         $objects = scandir($dir);
140         foreach ($objects as $object) {
141             if ($object != "." && $object != "..") {
142                 if (is_dir($dir . DIRECTORY_SEPARATOR . $object) && !is_link($dir . "/" . $object)) rmdir($dir . DIRECTORY_SEPARATOR . $object);
143                 else unlink($dir . DIRECTORY_SEPARATOR . $object);
144             }
145         }
146         rmdir($dir);
147     }
148 }
```

3.2.3 Övrigt

3.2.3.1 ASYNC-AWAIT

Asynkrona funktioner används vid anrop som kan ta tid, en vanlig användning är vid dataanrop från externa källor såsom en databas. Anledningen till att man använder *await* är att man behöver vänta på ett svar. Med nyckelordet *async* kan kod som är oberoende av svaret fortsättas att köras, vilket ger ett mer effektivt program. Viktigare än effektiviteten är att, när man definierar en variabel som sedan väntar på ett svar, så skulle efterkommande kod köras med den tomma variabeln om man inte angav att man väntar på ett svar. Om den efterkommande koden då är beroende av att variabeln ska innehålla ett värde kan stora problem inträffa.

3.2.3.2 FETCH

Fetch används för att hämta material. Det kan vara kompletta HTML-sidor men även mindre mallar. Under användningen av mallar i Firebaseprojektet användes den egengjorda funktionen *fetchPage* nedan för att hämta olika mallar och konvertera dem till en tjusterbar sträng. *Fetch*-funktionen är inte bunden endast till html-dokument, utan kan användas till att hämta i princip vad som helst, exempelvis javascript-kod.

Bilaga 7.24

```
async function fetchPage(path) {
    try {
        let response = await fetch(path); // returns promise
        return response.text();
    } catch (err) {
        console.log('Fetch error:' + err);
    }
}
```

3.2.3.3 TEMPLATE

Templates, eller mallar som det heter på svenska används för att skriva ut data på ett särskilt sätt. Mallar som används kan vara definierade direkt som en sträng i koden, men vid användning av flera mallar blir koden mer läsbar och utvecklarvänlig om man definierar mallarna i ett separat dokument. I PHP kan man enkelt importera en mall genom funktionen `file_get_contents`. JavaScripts motsvarighet till `file_get_contents` är `fetch`. Dock, med `fetch` hämtas inte en direkt förhandlingsbar sträng, utan den behöver konverteras till text vilket sköts i funktionen ovan, under Fetch-rubriken.

Vanligtvis ligger det särskilda taggar i mallarna som man byter ut med bland annat `string_replace`. Taggarna kan exempelvis vara `'{{title}}'` och det är något som användaren inte är menad att se.

3.3 SÄKERHET

Säkerhet är viktigt inom alla typer av projekt, oavsett om det är en webbsida eller om det är ett lokalt program, så kräver allt säkerhetshantering. Både säkerhet som förhindrar skadegörelse, och säkerhet som sköter felhantering, för att förhindra programkrascher och skydda användaren mot konstiga fel som kan konfundera dem.

3.3.1 Säkerhetsanordningar

3.3.1.1 AUTENTISERING

Båda webbprojekten tar säkerhet i hänsyn, dels för att det kan skada andra, dels för att det kan oss själva.

När det kommer till datahantering, kräver båda projekten att användaren ska vara inloggad för de ska ha möjlighet att skapa, uppdatera eller för att ta bort information. Båda projekten kräver det, men de sköter det på olika sätt.

I PHP-projektet krävs som sagt att användaren är inloggad. Verifieringen per sig är väldigt simpel, man kollar i sessionsvariabeln där man under inloggning (se Bilaga 7.10 rad 26) lägger in att värdet `loggedIn = true`. Därefter kan man kontrollera värdet, se bilagan under denna paragraf. Anledningen varför det är ett säkert sätt att hantera inloggning på är eftersom PHP sker på serversidan. Klienten, eller användaren har inte tillgång till dessa variabler och kan därav inte håller ändra på dem. Eftersom man kontrollerar inloggningsstatusen via en if-sats måste man kontrollera detta inför alla redigerings-funktioner.

Bilaga 7.25

```
if($_SESSION['loggedIn'] === true){
```

Firebaseprojektet sköter inloggningsverifiering på ett lite annorlunda sätt. Till att börja med, Firebases databas, Firestore har en egen regel-konsol där man kan sätta upp olika krav. I Bilagan 7.26 syns 3 rader som utgör grundreglerna för Firebase-projektet. Första raden, *allow read: if true* säger bara att alla har tillgång till att läsa information. Andra raden *allow delete if: ...* säger att *request.auth* måste vara skilt från *null*, med andra ord, användaren måste vara inloggad. Men *delete* har ett till krav, resursen eller

informationen som är skapad, den har en tag med ett unikt id, detta id:t måste matcha användarens id, den användaren som vill ta bort informationen. Tredje raden anger precis som början av rad två, att användaren måste vara inloggad, för att skapa eller uppdatera ett material. Enda nackdelen med Firestore-reglerna är att, det är ett nytt sätt att skriva på. Vet man inte hur reglerna ska skriva måste man söka upp vilket i sin tur kan göra det svårare att implementera egna regler. Att skriva egna regler kommer analyseras i under Auktorisering längre ner.

Bilaga 7.26

```
allow read: if true
allow delete: if request.auth != null && resource.data.uid == request.auth.uid
allow create, update: if request.auth != null
```

Autentisering kan även användas utanför datahantering, exempelvis för att visa eller dölja 'logga in' och logga ut' knappar. I PHP sköts det med samma if-stats som för datahanteringen. Det kräver att sidan laddas om för att den ska uppdateras, men med PHP skickas man mellan olika PHP-filer så det är inget användaren behöver sköta manuellt. Firebase däremot använder en eventlyssnare *onAuthStateChanged* där *firebaseUser* i bilagan nedan är användaren. Om användaren inte är inloggad är *firebaseUser* = *null*.

Bilaga 7.27

```
firebase.auth().onAuthStateChanged(async (firebaseUser) => {
```

3.3.1.2 KLIENT-SÄKERHET

En av de största säkerhetsriskerna i en webbapplikation är XSS, eller cross-site-scripting. Det innebär att man injekterar en bit kod men en specifik funktion, som är osynlig för den vardagliga användaren. Exakt vad koden gör kan variera kraftig beroende på vad injekterarens mål är. Vanligast är att man skickar användarens data, såsom *cookies* eller lyssnar på alla dess knapptryckningar för att få lösenord för att sedan skicka det till en extern server som sparar och använder den informationen. I PHP kan det skötas på flera sätt, bland annat genom att ersätta och lägga till karaktärer som bryter av den skadliga koden, eller att omvandla potentiella skadliga karaktärer till en sträng som motsvara den visuella karaktären. I Bilaga 7.13, PHP:s skaparfunktion sker detta på rad 12 till 14. Exempelvis kan öppnar taggen '<' göras om till '<', när webbläsaren ser den omgjorda texten visar den en motsvarighet till originala symbolen utan att köra potentiell kod inuti. Problemet med denna typ av säkerhetshantering kan vara radbyten, som i html-kod ser ut som följande,
. Byter man ut öppnings och stängningsstaggen fungerar inte radbrytningen och man behöver skriva ytterligare kod för lösa dessa nyskapta problem.

Firestore sköter dessa lösningar automatiskt åt programmeraren, dock saknas uppenbar dokumentation för exakt hur firebase löser denna säkerhetsrisk. Att man inte behöver skriva egna säkerhetsfunktioner gör att skapandet av en hemsida går betydligt snabbare och effektivare.

3.3.1.3 AUKTORISERING

Auktorisering, eller roller är endast insatt i Firebaseprojektet. Men implementeringen för det i PHP är ganska enkel. Tillvägagången för att ge ut administratörsprivilegier är genom att redigera användarfilen manuellt, vilket fungerar likadant i båda projekten.

Om man skulle införa roller i PHP skulle man i samband med autentiseringen kolla efter en potentiell roll, vilket visas i bilagan nedan. Enligt bilagan skulle den första satsen kräva att användaren är inloggad och den andra satsen kräver att användaren är administratör. Rollen som i exemplet ligger i sessionsvariabeln, som skulle läggas in under inloggningsprocessen med en snabb check om användaren har administratörsprivilegierna.

Bilaga 7.28

```
if($_SESSION['loggedIn'] === true && $_SESSION['role'] === 'admin'){
```

I Firebase har det varit större problem att implementera roller. För att ge en användare en roll måste man först kunna lagra den specifika rollen. Vilket har tidigare diskuterats under punkt 3.2.1.3 *Användarinformation*. Det innebär att en användare måste ange sitt användarnamn efter sin inloggning för att skapa sitt separata användardokument, vilket går för användaren att undkomma, vilket i sin tur innebär att man inte kan garantera möjligheten att ge användare administratörsprivilegierna. Ska en administratörsroll ges har användaren troligtvis redan loggat in, men om övriga roller ska ges kan det ställa till besvär. Utöver tidigare diskuterade nackdelar måste en inläsning av det separata användardokumentet ske varje gång en funktion använder särskilda roll-privilegier, vilket ytterligare i sin tur ökar datakonsumtionen, vilket i sin tur gör att storskaliga projekt kostar mer. Fördelen med Firebase är att när rollgivningen är färdig behöver man bara lägga till en enda rad, vilket kan ses i bilagan nedan. Man hämtar ut användardatan från Firestore och verifierar om användarens roll är lika med *admin*. Bilagans funktion gör att administratörer kan ta bort alla inlagda dokument oavsett om de skapade dokumentet eller inte, vilket har införts som krav enligt bilaga 7.26.

3.3.1.3.1 Bilaga 7.29

```
allow delete: if get(/databases/${database}/documents/users/${request.auth.uid}).data.role == 'admin'
```

Det stora problemet vid implementeringen av ovanstående sats är att Firestore-reglerna saknar lämpliga felsöknings-verktyg. Man kan inte använda några debug-verktyg för att kontrollera inmatningsvärden. Det innebär att när reglerna skrivs i regelkonsolen, så skrivs reglerna i det stora hela i blindo. Det går att kontrollera om resultatet fungerar genom att gå in på sin hemsida och testa om reglerna blockerar och släpper igenom rätt användare alternativt genom det inbyggda felsökningsverktyget *Rules Playground*. Däremot, om reglerna inte gör som man har avsett kommer de existerande verktygen inte att hjälpa då de i bästa fall bara anger att det är något som är fel. Det bästa man får göra är att jämföra sin kod med en eventuell guide eller att söka personlig hjälp, där det andra alternativet kan

vara svårt om man inte har en handledare, vilket i mitt fall krävdes för att rätta till satsen.

Utöver *Rules Playground* finns det även *Emulator suits* som bland annat kan köras på ens lokala dator, dock kräver det att projektet använder Java och Node.js vilket detta Firebaseprojekt inte handlar om. Därav är det inte ett användbart alternativ.

3.3.2 Felhantering

Felhantering är ett sätt att skydda och hjälpa användaren mot sig själv. Dels handlar det om att ge användaren logiska felmeddelanden, dels är det att undvika totaltrascher.

Exempelvis vid inloggningsprocessen eller registreringsprocessen får användaren olika meddelanden beroende på om dem inte har fyllt en mejl eller om de har angett ett felaktigt lösenord. Vilket visas under inloggningsanalysen tar Firebase och skickar ett fel som visas för användaren vid felaktig inloggning. Exakt hur texten som skickas är formulerad sköter Firebase. Men det finns fortfarande felhantering som behövs skrivas manuellt. I Firebaseprojektet som är ett bokbibliotek online, sköts individuell bokhämtning via get-variabler från URL:en, och har man angett fel eller en icke existerande bok behövs det förklaras för ens användare att boken inte kan hittas, vilket görs i bilaga 7.30. PHP-projektet felmeddelanden sköts av programmeraren. Fördelen med detta är att man har mer frihet angående vad som står i meddelandet medan nackdelen är att man måste skriva en del extrarader för att få ut bra felmeddelanden.

Utöver felmeddelanden vid inmatningsfällt pågår majoriteten av Firebase-funktionerna inom *try-catch*-block. Vilket gör att går något fel, kan man behandla felet på olika sätt, vanligast är att man använder *console.log* för att få ut ett felmeddelande.

Bilaga 7.30


```

//Makes sure book exist before doing anything else
async function checkBookIntegrity() {
  if (getQueryVariable('book') == false) {
    window.location.href = '/';
  } else {
    try {
      let docRef = await firebase
        .firestore()
        .collection('books')
        .doc(getQueryVariable('book'));

      docRef.get().then((doc) => {
        if (!doc.exists) {
          document.body.innerHTML = `
            <h2> Error 404: Book not found</h2>`;
        }
      });
    } catch (error) {
      console.log(error);
    }
  }
}

```

3.3.3 IP/Domän-bindning för Firebase

Inom firebase har man som tidigare förklarat en konfigurationsnyckel vilket tillåter ens hemsida att hämta och redigera information från ens Firebaseprojekt. Alla som har åtkomst till denna konfigurationsnyckel kan därav använda och redigera ens Firebaseprojekt, vilket är i sig en säkerhetsrisk. För att undkomma säkerhetsproblemet kan man använda sig av en tjänst som kallas för Firebase Hosting, där man bland annat får möjligheten att binda sin konfigurationsnyckel till en domän. Detta innebär att den angivna domänen, och endast den domänen har åtkomst till att läsa och ändra ens Firebaseprojekt.

Inom Firebaseprojektet som analyseras har denna funktion inte införts. Det beror på att Firebase Hosting är en betaltjänst, och därav har användningen av denna tjänst uteslutits från Firebaseprojektet.

4 DISKUSSION OCH SLUTSATS

Efter detta projekt är det klart att både Firebase och PHP har många fördelar, men båda har även sina nackdelar.

Många av fördelarna med Firebase är att alla delar som är jobbiga att genomföra; datalagring, säkerhet, autentisering och auktorisering görs väldigt smidigt. Jämförs längden kod mellan projekten är delar av Firebases funktioner bara ett par rader mot PHP:s sidlängder med kod. Det gör att uppstarten av Firebase är väldigt smidig och snabb. Men vilket som har framkommit så är det på en bekostnad av flexibilitet, vilket PHP klart vann i. För att lägga till användarnamn eller administratörsprivilegier var PHP bara en extra kodrad medan Firebase krävde en betydligt större tidsinsats, särskilt då Firebase-reglerna helt saknar felsökningsmetoder, men Firebase krävde knepiga och mindre snygga omvägar som i sin tur medför extra datatrafik för att uppfylla dessa mål. PHP:s fördelar gentemot Firebase är att man vet hur allting fungerar och att man på gott och ont har större kontroll över sitt projekt. Fler fördelar med PHP är att PHP är helt gratis att använda, om man bortser från webbserverkostnader för båda projekten är PHP helt gratis gentemot Firebases funktioner som kommer att kosta pengar vid ökad användning för att inte nämna att Firebase kostar pengar vid önskan att binda sin konfigurationsnyckel till en domän.

Genom att lära mig hur Firebase fungerar har jag fått stora kunskaper inom JavaScript, vilket jag oberoende av min framtida användning av Firebase kommer ha stor nytta utav. Firebase har även givit mig en utökad förståelse för databasanvändning och programmering i övrigt. Får jag frågan om det var värt att lära sig Firebase skulle jag i slutändan svara, ja. Eftersom det har gett mig så stora kunskaper inom programmering som helhet.

När man kommer komma till valet mellan Firebase och PHP i framtida bruk kommer valet baseras på vad projektets syfte är. Om man söker en snabb uppstart med mindre mängd datalagring som inte kommer ha stor trafik är Firebase utmärkt då auktorisering, datalagring och säkerhet sköts automatiskt om det inte vore för att Domän/IP-Bindning är en betaltjänst vilket medför säkerhetsrisker om den tjänsten inte tillköps. Men om projektet kan få mer än minimal trafik och om jag föredrar flexibel frihet, kommer jag välja PHP. För att göra PHP användningen mycket smidigare kommer dock en databas troligtvis att införas då man slipper sköta ett eget filsystem vilket PHP-projektet behövde göra, därav krävdes en betydligt längre kod vid CRUD-funktionerna.

5 KÄLL- OCH LITTERATURFÖRTECKNING

Stevenson. D. (2018, 25 sep). What is Firebase? The complete story, abridged. *Medium.com*. Tillgänglig från <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>

Clark. J. (u.å). What is Firebase Authentication. *Back4App.com*. Tillgänglig från <https://blog.back4app.com/firebase-authentication/>

Do. L. (2020, 9 jan). Implement Authentication in Flutter with Firebase Authentication. *Medium.com*. Tillgänglig från <https://medium.com/@dotronglong/implement-authentication-in-flutter-with-firebase-authentication-4cf0a133c488>

6 BILAGOR

6.1 FIREBASE - JAVASCRIPT

main.js

http://localhost:4649/?mode=javascript

```
const db = firebase.firestore(); // grants access firestore-database

fbUser = firebase.auth().currentUser; // FireBookListUser

const bookCol = db.collection('books');

const DQ = document.querySelector.bind(document);

firebase.auth().onAuthStateChanged(async (firebaseUser) => {
  try {
    // If user is logged in render and remove some stuff
    if (firebaseUser) {
      customRender(
        'input',
        'button',
        'Add Book to Library',
        'openBookForm',
        '#addContainer'
      );
      DQ('#openBookForm').addEventListener('click', renderAddBook);

      // try {
      /*
      return console.log(`!! DEV-MANUAL-WARNING !!
      Intentionallt cutting out rest of function and returning console.log! This is on
line ${
      Error().lineNumber
    } in main.js.
    This keeps unfixed errors from firing!
    !/ DEV-MANUAL-WARNING !/`); /* */
      console.log(firebaseUser.uid);
      let raw = await db.collection('users').doc(firebaseUser.uid).get();

      // Render username (and role) to welcomescreen

      if (await raw.data()) {
        let username = await raw.data().username;

        document.getElementById('username').innerHTML =
          'Welcome to The Book List ' + username;

        if (raw.data().role) {
          document.getElementById(
            'username'
          ).innerHTML += ` , <br> Your privilages are: ${await raw.data().role}`;
        }
      }

      // } catch (err) {
      //   console.log(err.message);
      // }
    } else {
      // If user is NOT logged in render and remove some stuff
```

1 of 2

2021-04-11, 15:45

```

54     if (DQ('#openBookForm')) removeElement('#openBookForm');
55
56     document.getElementById('username').innerText = 'Please log in!';
57   }
58 } catch (error) {
59   console.log(error);
60 }
61 });
62
63 ///!! STOP DO NOT WORK WITH ROLES OR USERNAMES JUST GET SHIT DONE
64
65 ///!!!!!! -----
66 /* TODO Add an eventlistener on the buttons which already exist, ie body onload.
67 The let those listeners look for a parameter when the button are prestet and send 'this'
   into the correct fuction.
68 Else create / destroy ecentlisteners when creating buttons
69 */
70 ///!!!!!! -----

```

```

// could use regExp, but you will solve one problem in order to get two new ones

// gets a html-page or whatever you want to fetch
async function fetchPage(path) {
  try {
    let response = await fetch(path); // returns promise
    return response.text();
  } catch (err) {
    console.log('Fetch error:' + err);
  }
}

// renders loginform and appends to body
async function renderLogin() {
  var loginDiv = document.createElement('div');
  loginDiv.innerHTML = await fetchPage('./html/loginForm.html');

  loginDiv.style = `background-color: rgba(100, 100, 100, 0.8);
margin: auto;
width: 100vw;
height: 100vh;
position: fixed;
top: 0;
z-index: 5;
display: flex;
justify-content: center;`;
  loginDiv.id = 'loginId';

  let resp = await document.body.appendChild(loginDiv);

  console.log('print to screen Login');

  DQ('#formLogin').addEventListener('submit', loginUser);

  DQ('#loginId').addEventListener('click', (event) => {
    if (event.target.id == 'loginId') removeElement('#loginId');
  });
}

// renders registerform and appends to body
async function renderRegister() {
  var registerDiv = document.createElement('div');
  registerDiv.innerHTML = await fetchPage('./html/registerForm.html');

  registerDiv.style = `background-color: rgba(100, 100, 100, 0.8);
margin: auto;
width: 100vw;
height: 100vh;
position: fixed;
top: 0;
z-index: 5;
display: flex;
justify-content: center;`;
  registerDiv.id = 'registerId';
}

```

```

55
56 let resp = await document.body.appendChild(registerDiv);
57 console.log('print to screen Register');
58 document
59   .querySelector('#formRegister')
60   .addEventListener('submit', registerUserInFirebase);
61 DQ('#registerId').addEventListener('click', (event) => {
62   if (event.target.id === 'registerId') removeElement('#registerId');
63 });
64 }
65
66 // renders addbook-form and appends to body
67 async function renderAddBook() {
68   var bookAdderDiv = document.createElement('div');
69   bookAdderDiv.innerHTML = await fetchPage('./html/AddBook.html');
70
71   bookAdderDiv.style = `background-color: rgba(100, 100, 100, 0.8);
72   margin: auto;
73   width: 100vw;
74   height: 100vh;
75   position: fixed;
76   top: 0;
77   z-index: 5;
78   display: flex;
79   justify-content: center;`;
80   bookAdderDiv.id = 'bookAdderId';
81
82   let resp = await document.body.appendChild(bookAdderDiv);
83   console.log('print to screen AddBook');
84   document
85     .querySelector('#formAddBook')
86     .addEventListener('submit', addBookToDataBase);
87   DQ('#bookAdderId').addEventListener('click', (event) => {
88     if (event.target.id === 'bookAdderId') removeElement('#bookAdderId');
89   });
90 }
91
92 // renders updateform and appends to body
93 async function renderUpdateForm(bookId) {
94   var bookUpdateDiv = document.createElement('div');
95   bookUpdateDiv.innerHTML = await fetchPage('./html/updateForm.html');
96
97   bookUpdateDiv.style = `background-color: rgba(100, 100, 100, 0.8);
98   margin: auto;
99   width: 100vw;
100  height: 100vh;
101  position: fixed;
102  top: 0;
103  z-index: 5;
104  display: flex;
105  justify-content: center;`;
106   bookUpdateDiv.id = 'bookUpdateId';
107
108   let resp = await document.body.appendChild(bookUpdateDiv);
109   console.log('print to screen AddBook');

```



```

110 DQ('#updId').value = bookId;
111 DQ('#formUpdate').addEventListener('submit', updateBooks);
112 DQ('#bookUpdateId').addEventListener('click', (event) => {
113     if (event.target.id == 'bookUpdateId') removeElement('#bookUpdateId');
114 });
115 }
116 // ! Doesnt need to fetch navBar, I'll use different ones regardless
117 /*
118
119 renderNavigationBar();
120
121 async function renderNavigationBar() {
122     var navigation = document.createElement('nav');
123     console.log(window.location.pathname);
124     if (window.location.pathname == '/bookReader.html') {
125         navigation.innerHTML = await fetchPage('./html/navigationbarBooks.html');
126     } else {
127         navigation.innerHTML = await fetchPage('./html/navigationbar.html');
128     }
129     /* navigation.innerHTML = `
130     <ul>
131     <li class="li1"><button onclick="location.href='/'">Home</button></li>
132     <li class="li2"><h1>My Book List</h1></li>
133     <li class="li3"><button
134         onclick="location.href='{logoutPath}'">Logout</button></li>
135     </ul>
136     `; */
137     /* navigation.id = 'navigationbar';
138     console.log(await navigation);
139     resp = await document.querySelector('#navbar').appendChild(navigation);
140     console.log('Navbar:', resp);
141 }*/
142
143 /*
144 function renderSingout() {
145     console.log('Render Singout');
146
147     var singout = document.createElement('input');
148     singout.type = 'button';
149     singout.value = 'Sing out';
150     singout.id = 'signOut';
151     document.getElementById('userAuthControls').appendChild(singout);
152
153     document.querySelector('#signOut').addEventListener('click', () => {
154         firebase.auth().signOut();
155     });
156 }*/
157
158 async function renderUsernameRequest() {
159     var bookUpdateDiv = document.createElement('div');
160     bookUpdateDiv.innerHTML = await fetchPage('./html/requestUsernameForm.html');
161
162     bookUpdateDiv.style = `background-color: rgba(100, 100, 100, 0.8);
163     margin: auto;

```

```
164 width: 100vw;
165 height: 100vh;
166 position: fixed;
167 top: 0;
168 z-index: 5;
169 display: flex;
170 justify-content: center;`;
171 bookUpdateDiv.id = 'usernameId';
172
173 let resp = await document.body.appendChild(bookUpdateDiv);
174 console.log('print to screen get username');
175 DQ('#formUsername').addEventListener('submit', addUsernameToUsercollection);
176 DQ('#usernameId').addEventListener('click', (event) => {
177   if (event.target.id == 'usernameId') removeElement('#usernameId');
178 });
179 }
180
181 // Allows custom render of elements, meant for input-fields
182 function customRender(element, type = null, value, id, addTo) {
183   var createElement = document.createElement(element);
184   if (type) createElement.type = type;
185   createElement.value = value;
186   createElement.id = id;
187   document.querySelector(addTo).appendChild(createElement);
188 }
189
190 // removes parent of element, including queryselectors
191 function removeParrent(event) {
192   console.log(event.parentElement.id);
193   document.querySelector(event.parentElement.id).remove();
194   console.log('Removed:', event.parentElement.id);
195 }
196
197 // removes element, you can send it as an object or send an id directly, including
  removal of queryselectors
198 function removeElement(event) {
199   document.querySelector(event.id || event).remove();
200   console.log('Removed:', event.id || event);
201 }
---
```



```

firebase.auth().onAuthStateChanged(async (firebaseUser) => {
  try {
    if (firebaseUser) {
      console.log(firebaseUser.uid);

      // Removes possible loginFrom
      if (DQ('#formLogin') !== null) {
        removeElement('#formLogin');
      }
      if (DQ('#registerId') !== null) {
        removeElement('#registerId');
      }

      // Renders Sign Out button
      customRender('input', 'button', 'Sign Out', 'signOut', '#singOutButton');
      DQ('#signOut').addEventListener('click', () => {
        firebase.auth().signOut();
      });

      // Removes login/register
      // short hand if-statements, DQ = doc-.querySelector()
      if (DQ('#getLogin')) DQ('#getLogin').remove();
      if (DQ('#getRegister')) DQ('#getRegister').remove();

      // If user does not have a username, add it
      let raw = await db.collection('users').doc(firebaseUser.uid).get();
      if (!(await raw.data())) {
        renderUsernameRequest();
      }
    } else {
      // Remove login button
      if (DQ('#signOut')) DQ('#signOut').remove();
      if (DQ('#openBookForm')) DQ('#openBookForm').remove();

      // Generates LoginButton
      customRender(
        'input',
        'button',
        'Log In',
        'getLogin',
        '#userAuthControls'
      );

      DQ('#getLogin').addEventListener('click', renderLogin);

      // Generates Registerbutton
      customRender(
        'input',
        'button',
        'Register',
        'getRegister',
        '#userAuthControls'
      );
    }
  }
});

```

```
55     DQ('#getRegister').addEventListener('click', renderRegister);
56   }
57 } catch (error) {
58   console.log(error);
59 }
60 });
61
62 // Log in the user
63 function loginUser(event) {
64   try {
65     event.preventDefault();
66     console.log(this);
67
68     const email = this.email.value;
69     const pwd = this.pwd.value;
70
71     const promise = firebase.auth().signInWithEmailAndPassword(email, pwd);
72     promise.catch((e) => alert(e.message));
73
74     removeElement('#loginId');
75   } catch (error) {
76     console.log(error);
77   }
78 }
79
80 //registers and adds username
81 async function registerUserInFirebase(event) {
82   try {
83     event.preventDefault();
84     if (firebase.auth().currentUser == null) {
85       console.log(
86         'Registering when not logged in',
87         firebase.auth().currentUser
88       ); // expected null
89       console.log(this);
90       const email = this.email.value;
91       const pwd = this.pwd.value;
92       const promise = firebase
93         .auth()
94         .createUserWithEmailAndPassword(email, pwd);
95       promise.catch((e) => alert(e.message));
96       if (!promise.catch()) removeElement('#registerId');
97       //firebase baksida, du har inte tillgång till auth tabellen
98     } else {
99       console.log('User cannot register when signed in!');
100     }
101   } catch (error) {
102     console.log(error);
103   }
104 }
105
106 // Adds username to collection user
107 async function addUsernameToUsercollection(event) {
108   try {
109     event.preventDefault();
```

```
110   if (this.usernameField.value) {
111     let newUser = { username: this.usernameField.value };
112
113     await db
114       .collection('users')
115       .doc(firebase.auth().currentUser.uid)
116       .set(newUser);
117     removeElement('#usernameId');
118   } else {
119     alert('Username is empty! Make sure to fill out all fields!');
120   }
121 } catch (error) {
122   console.log(error);
123 }
124 }
125 // Get a users username by thier id
126 async function getUsernameById(uid) {
127   console.log(uid);
128   let raw = await db.collection('users').doc(uid).get();
129
130   // Render usernamne (and role) to welcomescreen
131
132   if (await raw.data()) {
133     return await raw.data().username;
134   }
135 }
```

```

checkBookIntegrity();

console.log('Book-id:', getQueryVariable('book'));

const db = firebase.firestore(); // grants access firestore-database
const DQ = document.querySelector.bind(document); // Can now write
doc--.querySelector(-- as DQ(--))
const bookId = getQueryVariable('book');

// Loads bookdata
// getBookData();

//Makes sure book exist before doing anything else
async function checkBookIntegrity() {
  if (getQueryVariable('book') == false) {
    window.location.href = '/';
  } else {
    try {
      let docRef = await firebase
        .firestore()
        .collection('books')
        .doc(getQueryVariable('book'));

      docRef.get().then((doc) => {
        if (!doc.exists) {
          document.body.innerHTML = `
            <h2> Error 404: Book not found</h2>`;
        }
      });
    } catch (error) {
      console.log(error);
    }
  }
}

// Loads and unloads stuff depending on login-status
firebase.auth().onAuthStateChanged((firebaseUser) => {
  if (firebaseUser) {
    var updButton = `<button id="bookUpdater"
onClick="renderUpdateForm('${bookId}')">Update book</button>`;
    document.querySelector('#bookDoer').innerHTML = updButton;
  } else {
    if (DQ('#bookUpdater')) removeElement('#bookUpdater');
  }
});

// Renders book info to page
db.collection('books')
.doc(bookId)
.onSnapshot(async (doc) => {
  book = await doc;
  console.log(await book.data());
  renderTitle(await book.data());
});

```

```

53   renderInfo(await book.data());
54   renderSynopsis(await book.data());
55   findOtherBooksInSeries(await book.data());
56   renderCreatorAndEditors(await book.data());
57   });
58
59   // gets url get-var
60   function getQueryVariable(variable) {
61     var query = window.location.search.substring(1);
62     var vars = query.split('&');
63     for (var i = 0; i < vars.length; i++) {
64       var pair = vars[i].split('=');
65       if (pair[0] == variable) {
66         return pair[1];
67       }
68     }
69     return false;
70   }
71
72   async function renderTitle(book) {
73     // Print title & serie
74
75     var item = document.createElement('div');
76
77     item.innerHTML = `<h2>${await book.title}</h2><br>`;
78
79     if (await book.serie) item.innerHTML += `<h3><i>${await book.serie}</i></h3>`;
80
81     item.id = 'bookTitle';
82
83     DQ('#bookTitle')
84     ? (DQ('#bookTitle').innerHTML = item.innerHTML)
85     : DQ('#title').appendChild(item);
86   }
87
88   async function renderInfo(book) {
89     // Print Author, realese and pages
90     var item = document.createElement('div');
91     item.id = 'bookInfo';
92
93     if (book.author) {
94       item.innerHTML += `<div> Written by: ${book.author}</div><br>`;
95     }
96     if (book.year) {
97       item.innerHTML += `<div> Release year: ${book.year}</div><br>`;
98     }
99     if (book.pages) {
100      item.innerHTML += `<div> Amount of pages: ${book.pages}</div><br>`;
101    }
102    if (item != '') {
103      DQ('#bookInfo')
104      ? (DQ('#bookInfo').innerHTML = item.innerHTML)
105      : DQ('#info').appendChild(item);
106    }
107  }

```



```

108
109 async function renderSynopsis(book) {
110   var item = document.createElement('div');
111
112   // Print synopsis
113   console.log(await book.synopsis);
114   item.id = 'bookSynopsis';
115   if (await book.synopsis) {
116     item.innerHTML = `<p>${await book.synopsis}</p>`;
117   } else {
118     item.innerHTML = `
119     <p><i>
120     This book seems to lack a synopsis.
121     If you know any information about it, please
122     update it to complete our database!</i></p>`;
123   }
124   item.id = 'bookSynopsis';
125
126   //DQ('#synopsis').appendChild(item);
127
128   console.log('bookSynopsis Exists:', DQ('#bookSynopsis') !== undefined);
129   if (DQ('#bookSynopsis') !== undefined) {
130     DQ('#bookSynopsis').innerHTML = item.innerHTML;
131   } else {
132     DQ('#synopsis').appendChild(item);
133   }
134 }
135
136 async function findOtherBooksInSerie(book) {
137   // Looks and checks weather there are other books in the same serie
138   var books = [];
139
140   const booklister = firebase.firestore().collection('books');
141
142   booklister
143     .where('searchSerieQuery', '==', book.searchSerieQuery)
144     .orderBy('year', 'asc')
145     .onSnapshot((docs) => {
146       var books = [];
147       booksUnfindend = [];
148
149       // docChanges() gets changing documents, snapshot gets changed collection
150       docs.docChanges().forEach((element) => {
151         if (book.title !== element.doc.data().title && element.doc.data().year) {
152           books.push({ id: element.doc.id, data: element.doc.data() });
153         } else if (book.title !== element.doc.data().title) {
154           booksUnfindend.push({ id: element.doc.id, data: element.doc.data() });
155         }
156       });
157       console.log(books);
158       if (books.length !== 0) {
159         console.log(books);
160         DQ(
161           '#otherBooks'
162         ).innerHTML = `<h3>Other books in the serie: ${book.serie}</h3><hr><br></div>`

```

```

    };
    163     renderOtherBooksInSeries(books, book);
    164     DQ('#otherBooks').innerHTML += `<br>`;
    165     renderOtherBooksInSeries(booksUndeified, book);
    166   } else {
    167     DQ(
    168       '#otherBooks'
    169     ).innerHTML = `<h3>Other books in serie; ${book.series}</h3><br><i> There are n
other books in this serie </i>`;
    170   }
    171   });
    172 }
    173
    174 async function renderOtherBooksInSeries(info, book) {
    175   let output = info.map((i) => {
    176     if (i.data.year > book.year && i.data.year) {
    177       return `
    178       <a href="bookReader.html?book=${i.id}" id="${i.id}">
    179       <h3><i>Sequel</i>: ${i.data.title}</h3></a>
    180       `;
    181     } else if (i.data.year < book.year && i.data.year) {
    182       return `
    183       <a href="bookReader.html?book=${i.id}" id="${i.id}">
    184       <h3><i>Prequel</i>: ${i.data.title}</h3></a>
    185       `;
    186     } else {
    187       return `
    188       <a href="bookReader.html?book=${i.id}" id="${i.id}">
    189       <h3> ${i.data.title}</h3></a>
    190       `;
    191     }
    192   });
    193   document.querySelector('#otherBooks').innerHTML += output.join('<br>');
    194 }
    195
    196 async function renderCreatorAndEditors(book) {
    197   console.log('editors', book.editors);
    198   let output = `Creator: ${await getUsernameById(book.uid)} <br>`;
    199   let outputEditors = `Editors: `;
    200   for (var key in book.editors) {
    201     if (book.editors.hasOwnProperty(key)) {
    202       outputEditors += ` ${await getUsernameById(key)}, `;
    203     }
    204   }
    205
    206   DQ('#creatorAndEditor').innerHTML = `<h3>Creator And Editors</h3><hr>`;
    207   DQ('#creatorAndEditor').innerHTML += output;
    208   if (outputEditors !== `Editors: `) {
    209     DQ('#creatorAndEditor').innerHTML += outputEditors;
    210   }
    211   /*
    212   let outputs = book.editors.map((i, time) => {
    213     if (i.data.year > book.year && i.data.year) {
    214       return `
    215       <a href="bookReader.html?book=${i.id}" id="${i.id}">

```

```

    216     <h3><i>Sequel</i>: ${i.data.title}</h3></a>
    217     `;
    218   } else if (i.data.year < book.year && i.data.year) {
    219     return `
    220     <a href="bookReader.html?book=${i.id}" id="${i.id}">
    221     <h3><i>Prequel</i>: ${i.data.title}</h3></a>
    222     `;
    223   } else {
    224     return `
    225     <a href="bookReader.html?book=${i.id}" id="${i.id}">
    226     <h3> ${i.data.title}</h3></a>
    227     `;
    228   }
    229   });
    230   document.querySelector('#otherBooks').innerHTML += output.join('<br>'); */
    231 }

```

```

let init = false; // Used to determine book-load process

// Waiting for document to load before loading all books
document.addEventListener('DOMContentLoaded', (event) => {
  const app = firebase.app();
  console.log(app);
  const db = firebase.firestore();
  const bookLister = db.collection('books');

  /*
  first fires an onSnapshot to load all documents, then using docChanges to listen for
  changed documents, and thereby using less data when updating because onSnapshot
  returns the entire collection
  */
  if (init) {
    bookLister.onSnapshot((docs) => {
      var books = [];
      // docChanges() gets changing documents, snapshot gets changed collection
      docs.docChanges().forEach((element) => {
        books.push({ id: element.doc.id, data: element.doc.data() });
        console.log(books);
        replaceClientBook(books);
      });
    });
  } else {
    bookLister.orderBy('timestamp', 'desc').onSnapshot((querySnapshot) => {
      var books = [];
      querySnapshot.forEach((doc) => {
        books.push({ id: doc.id, data: doc.data() });
      });
      renderBookCollection(books);
      console.log('Current cities in CA: ', books.join(', '));
    });
    init = true;
  }
});

// Replace a single book, use with docChanges()
function replaceClientBook(info) {
  let bookId;
  let output = info.map((book) => {
    bookId = book.id;

    return `<a href="bookReader.html?book=${i.id}">
    <h3>${book.data.title}, ${book.data.series}</h3><br>
    <h3><i>${book.data.author}</i><br></a>
    <button onclick="renderUpdateForm('${book.id}')" > update </button>
    <button onclick="deleteDocs('${book.id}')" > Delete </button>
    `;
  });

  console.log('let bookId', bookId);
  document.getElementById(bookId).innerHTML = output;
}

```

```

55 }
56
57 // Renders intire book collection
58 function renderBookCollection(info) {
59   let output = info.map((i) => {
60     return `<a href="bookReader.html?book=${i.id}"><div id="${i.id}">
61       <h3>${i.data.title}, ${i.data.series} </h3><br>
62       <h3><i>${i.data.author}</i><br></a>
63       <button onclick="renderUpdateForm('${i.id}')" > update </button>
64       <button onclick="deleteDocs('${i.id}')" > Delete </button>
65     </div>
66     `;
67   });
68   document.querySelector('#allBooks').innerHTML = output.join('<hr>');
69 }

```


key.js

http://localhost:4649/?mode=javascript

```
1 // Your web app's Firebase configuration
2 var firebaseConfig = {
3   apiKey: 'AIzaSyBzwBhIkcsAH[redacted]f_Ij2U5gcQ',
4   authDomain: 'firebook-[redacted].firebaseapp.com',
5   projectId: 'firebook-[redacted]',
6   storageBucket: 'firebook-[redacted].appspot.com',
7   messagingSenderId: '9824[redacted]749',
8   appId: '1:9824[redacted]749:web:ae39b79[redacted]3ff5a69c',
9 };
10 // Initialize Firebase
11 firebase.initializeApp(firebaseConfig);
```

(Obs, delar av nyckeln är dold utav säkerhetsskäl)

6.2 FIREBASE – HTML, HTML-MALLAR & CSS

index.html

http://localhost:4649/?mode=htmlmixed

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!-- Gets firebase libraries -->
    <script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-app.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-firestore.js">
  </script>
    <script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-auth.js"></script>
    <!-- Gets FirebaseKey -->
    <script src="validation/key.js"></script>
    <!-- Renderfunctions -->
    <script src="render.js" defer></script>

    <!-- The documnt who does stuff -->
    <script src="main.js" defer></script>
    <script src="crud.js" defer></script>

    <!-- Handels everything related to auth -->
    <script src="authRel.js" defer></script>
    <!-- Get / Handle Books -->
    <script src="getBooks.js" defer></script>
    <!-- Gets stylesheet -->
    <link rel="stylesheet" href="homepage.css" />
    <title>FireBookList</title>
  </head>
  <body>
    <!-- Create login form here -->
    <!-- TODO Create div usefyll -->

    <div class="gridcontainer" i>
      <!-- use assoc array, with ['header']->titel etc -->

      <div class="sidel"><!-- empty --></div>

      <div class="maincontent">
        <div class="nav" id="navbar">
          <nav>
            <ul>
              <li class="li1">
                <button onclick="location.href='{homePath}'">Home</button>
              </li>
              <li class="li2"><h1>My Book List</h1></li>
              <li class="li3" id="singOutButton"></li>
            </ul>
          </nav>
        </div>
        <div class="header" style="display: flex">
          <!-- TODO: Put login, reg, auth here -->
          <div style="flex: 1" id="username"></div>
          <div id="userAuthControls" style="flex: 0; min-width: 78px">
            <!-- <input type="button" value="Login" id="getLogin" />

```

1 of 2

2021-04-11, 15:13

```

54         <input type="button" value="Register" id="getRegister" /> -->
55         <!-- TODO: Put login/ reg button here -->
56         <!-- <button onclick="location.href='{{}}'">Update Book</button>
57         <button onclick="location.href='{{deletePath}}'">Delete Book</button> -->
58     </div>
59     <!-- TODO: Sing out, not sing out -->
60 </div>
61 <div class="contentBox">
62     <div class="review">
63         <div style="display: flex; flex-direction: row">
64             <h3 style="flex: 1">Book Library</h3>
65             <span style="flex: 0" id="addContainer"> </span>
66         </div>
67
68         <hr />
69         <div id="allBooks"></div>
70         <br />
71     </div>
72
73     <div class="synopsis">
74         <h3>Latest Discussions</h3>
75         <hr />
76         <i>
77             >Yu-hue! <br />
78             Wanna start a forum and discuss a topic? Use reddit instead, this
79             is not implemeted!<br>
80         </i>
81     </div>
82 </div>
83
84 <div class="footer">
85     <h4 style="color: #333; text-align: center">
86         © 2020 - 2020 <i>My Book List </i> All Rigts Reserved
87     </h4>
88 </div>
89 </div>
90 <div class="sideR"><!-- empty --></div>
91 </div>
92 </body>
93 </html>

```

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!-- Gets firebase libraries -->
    <script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-app.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-firestore.js">
  </script>
    <script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-auth.js"></script>
    <!-- Gets FirebaseKey -->
    <script src="validation/key.js"></script>
    <!-- Renderfunctions -->
    <script src="render.js" defer></script>
    <script src="crud.js" defer></script>

    <!-- The documnt who does stuff -->
    <script src="authRel.js" defer></script>
    <script src="bookHandler.js" defer></script>

    <!-- Get / Handle Books -->
    <!-- <script src="getBooks.js" defer></script> -->
    <!-- Gets stylesheet -->
    <link rel="stylesheet" href="db/layouts/homepage/homepage.css" />
    <title>FireBookList</title>

  <body class="gridcontainer">

    <div class="sideL"><!-- empty --></div>

    <div class="maincontent">
      <div class="nav" id="navbar"><nav>
        <ul>
          <li class="li1">
            <button onclick="location.href='/'">Home</button>
          </li>
          <li class="li2"><h1>My Book List</h1></li>
          <li class="li3" id="singOutButton"></li>
          <li id="userAuthControls"></li>
        </ul>
      </nav></div>
      <div class="header" style="display: flex;">
        <div style="flex:1;" id="title"></div>
        <div style="flex:0; min-width:78px;" id="bookDoer"><!-- <button
onclick="location.href='{{updatePath}}'">Update Book</button>
<button onclick="location.href='{{deletePath}}'">Delete Book</button> -->
        </div>
      </div>
      <div class="contentBox" style="display: flex;">
        <div class="review" id="info" style="flex:0">
          <h3> Information</h3><hr>
        </div>

```

```

53
54
55     <div class="synopsis" id="synopsis" style="flex:1">
56         <h3>Synopsis</h3><hr>
57         <br>
58     </div>
59
60     <!-- <div class="review" ><h3> Reviews</h3><hr>
61
62     <button onclick="location.href='{{crReviewPath}}'">Create
review</button><hr>
63     <br>{{reviews}}<br>
64     </div>
65     -->
66     <div class="synopsis" id="otherBooks"></div>
67
68
69     <div class="synopsis" id="creatorAndEditor">
70         <h3>Creator And Editors</h3><hr>
71     </div>
72
73
74     <form action="addreview.html" method="post"></form>
75     </div>
76     <div class="footer">
77         <h4 style="color:#333; text-align:center;">© 2020 - 2020 <i>My Book List
</i> All Rights Reserved</h4>
78     </div>
79
80 </div>
81 <div class="sideR"><!-- empty --></div>
82
83
84 </body>
85
86 </html>

```

```
1 <button id="" style="height: 25px" onclick="removeElement('#loginId')">
2   Go back
3 </button>
4 <div
5   class="register"
6   style="margin: auto; padding: 1em; background-color: rgb(100, 100, 100)"
7 >
8   <h2>Login with FireBase Authentication</h2>
9   <form
10     id="formLogin"
11     style="
12       display: flex;
13       justify-content: center;
14       padding: 0.5em;
15       flex-direction: column;
16     "
17   >
18     <input type="text" placeholder="email" id="email" /><br />
19     <input type="password" placeholder="pwd" id="pwd" /><br />
20     <input type="submit" value="Login" id="btnLogin" />
21   </form>
22   <br />
23 </div>
```



```

1 <nav>
2   <ul>
3     <li class="li1">
4       <button onclick="location.href='{homePath}'">Home</button>
5     </li>
6     <li class="li2"><h1>My Book List</h1></li>
7     <li class="li3" id="singOutButton"></li>
8   </ul>
9 </nav>

```

```

1 <button id="" style="height: 25px" onclick="removeElement('#usernameId')">
2   Go back
3 </button>
4 <div
5   class="username"
6   style="margin: auto; padding: 1em; background-color: rgb(100, 100, 100)"
7 >
8   <h2>Add a username</h2>
9
10  <form
11    id="formUsername"
12    style="
13      display: flex;
14      justify-content: center;
15      padding: 0.5em;
16      flex-direction: column;
17    "
18  >
19    <input
20      type="text"
21      id="usernameField"
22      placeholder="YourAwesomeUsername"
23      required
24    /><br />
25
26    <input type="submit" value="Add a username" id="btnAddUsername" />
27    <i style="font-size: 0.4em; text-align: center"
28      >This action is permanent and cannot be changed</i
29    >
30  </form>
31 </div>

```

```

1 <button id="" style="height: 25px" onclick="removeElement('#registerId')">
2   Go back
3 </button>
4 <div
5   class="register"
6   style="margin: auto; padding: 1em; background-color: rgb(100, 100, 100)"
7 >
8   <h2>Register to thee Firebase Operational HQ</h2>
9
10  <form
11    id="formRegister"
12    style="
13      display: flex;
14      justify-content: center;
15      padding: 0.5em;
16      flex-direction: column;
17    "
18  >
19    <input type="text" name="email" placeholder="email" /><br />
20    <!-- TODO: Change type from text to email -->
21    <input type="password" name="pwd" placeholder="password" /><br />
22    <input type="submit" value="Register" id="btnRegister" />
23  </form>
24 </div>

```

```

1 <button id="" style="height: 25px" onclick="removeElement('#bookAdderId')">
2   Go back
3 </button>
4 <div
5   class="addBook"
6   style="margin: auto; padding: 1em; background-color: rgb(100, 100, 100)"
7 >
8   <h2>Add Book To Firebase</h2>
9
10  <form
11    id="formAddBook"
12    style="
13      display: flex;
14      justify-content: center;
15      padding: 0.5em;
16      flex-direction: column;
17    "
18  >
19    <input type="text" id="addTitle" placeholder="title" required /><br />
20    <input type="text" id="addSerie" placeholder="serie" /><br />
21    <input type="text" id="addAuthor" placeholder="author" /><br />
22    <input type="text" id="addYear" placeholder="year" /><br />
23    <!-- <input type="text" id="addSynopsis" placeholder="synopsis" /><br /> -->
24    <textarea
25      name="addSynopsis"
26      id="addSynopsis"
27      cols="30"
28      rows="10"
29    ></textarea>
30    <input type="submit" value="Add Book To Database" id="btnAddBook" />
31  </form>
32 </div>

```

updateForm.html

http://localhost:4649/?mode=htmlmixed

```

1 <button id="" style="height: 25px" onclick="removeElement('#bookUpdateId')">
2   Go back
3 </button>
4 <div
5   class="update"
6   style="margin: auto; padding: 1em; background-color: rgb(100, 100, 100)"
7 >
8   <h2>Update Book In Database</h2>
9
10  <form
11    id="formUpdate"
12    style="
13      display: flex;
14      justify-content: center;
15      padding: 0.5em;
16      flex-direction: column;
17    "
18  >
19    <input type="text" id="updTitle" placeholder="title" /><br />
20    <input type="text" id="updSerie" placeholder="serie" /><br />
21    <input type="text" id="updAuthor" placeholder="author" /><br />
22    <input type="text" id="updYear" placeholder="year" /><br />
23    <textarea
24      name="updSynopsis"
25      id="updSynopsis"
26      cols="30"
27      rows="10"
28    ></textarea>
29    <input type="hidden" id="updId" />
30    <input type="submit" value="Update Book To Database" id="btnUpdBook" />
31  </form>
32 </div>

```



```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

html {
  background-color: #ddd;
  min-height: 100%;
  margin: 0;
  padding: 0;
  display: flex;
}

body {
  min-height: 100%;
  margin: 0;
  padding: 0;
  background-color: #ffd4d4;
  flex: 1;
}

.flexcontainer {
  display: flex;
  background-color: #ffd4d4;
  width: 100%;
  height: 100%;
}

.flexbox {
  flex: 1;
  background-color: orange;
  width: 75%;
  height: 75%;
}

.gridcontainer {
  display: flex;
  margin: 0;
  padding: 0;
  flex: 1;
  min-height: 100%;
  background-color: #ffd4d4;
}

.sideL,
.sideR {
  flex: 5;
  background-color: #ffd4d4;
}

maincontent {
  display: flex;
  flex-direction: column;
  flex: 7;
  padding: 5px;
}
```

```
55 background-color: #fff;
56 border-width: 1px;
57 border-color: #ff8787;
58 border-style: solid;
59 min-height: 100%;
60 }
61
62 .nav {
63 height: 0fr;
64 flex: 0;
65 }
66 .header {
67 flex: 0;
68 height: 0fr;
69 padding: 5px;
70 margin: 25px;
71 margin-bottom: 10px;
72 background-color: #fff;
73 border-width: 1px;
74 border-color: #ff8787;
75 border-style: solid;
76 }
77
78 .synopsis {
79 flex: 0;
80
81 padding: 5px;
82 margin: 10px;
83
84 background-color: #fff;
85 border-width: 1px;
86 border-color: #ff8787;
87 border-style: solid;
88 }
89
90 .review {
91 flex: 5;
92
93 padding: 5px;
94 margin: 10px;
95
96 background-color: #fff;
97 border-width: 1px;
98 border-color: #ff8787;
99 border-style: solid;
100 }
101
102 .contentBox {
103 display: flex;
104 flex-direction: column;
105 flex: 1;
106 padding: 5px;
107 margin: 10px;
108
109 border-color: #ff8787;
```

```

110 }
111 .title > h3 {
112   color: #666;
113 }
114 .footer {
115   flex: 0;
116   padding: 5px;
117   margin: 25px;
118   margin-bottom: 0px;
119   min-height: 25px;
120   border-width: 1px;
121   border-color: gray;
122   border-style: solid;
123 }
124 }
125 hr {
126   padding: 0;
127   margin: 5px;
128 }
129 .review * hr {
130   color: #bbb;
131   background-color: #bbb;
132 }
133 }
134 a {
135   color: #000;
136   text-decoration: none;
137 }
138 a:visited {
139 }
140 a:hover {
141   text-decoration: underline;
142   text-decoration-color: #ff8787;
143 }
144 }
145 ul {
146   padding: 10px;
147   list-style: none;
148   display: flex;
149 }
150 .li1 {
151   flex: 0;
152   float: right;
153 }
154 .li2 {
155   flex: 1;
156   text-align: center;
157 }
158 .li3 {
159   flex: 0;
160   float: left;
161 }
162 /*
163 .contentBox * , .header *{
164   line-break: anywhere;

```

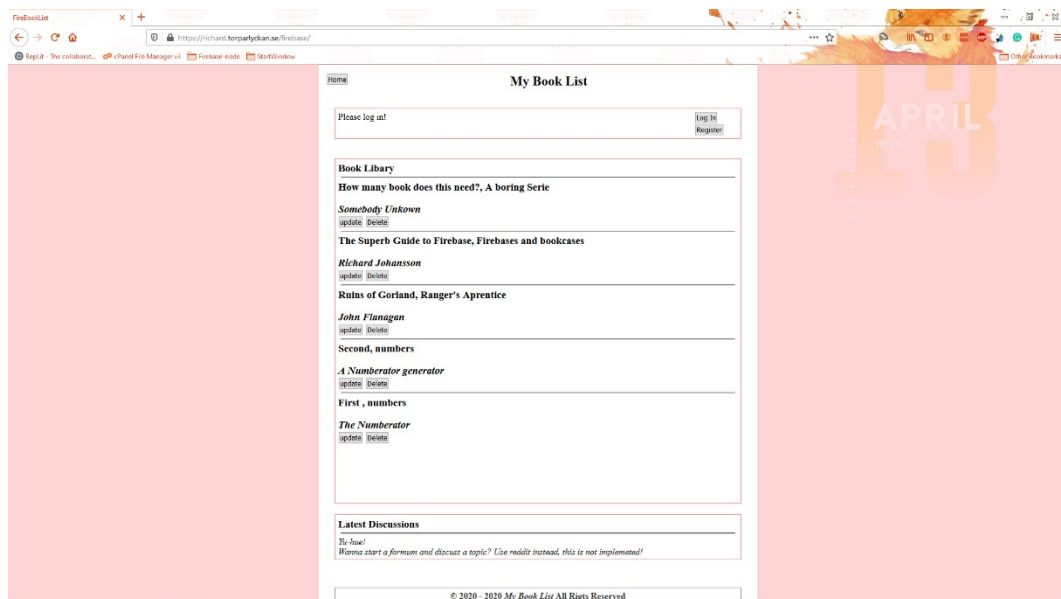
```

165 */
166 @media (max-width: 900px) {
167   .sideL,
168   .sideR {
169     flex: 0;
170   }
171 }

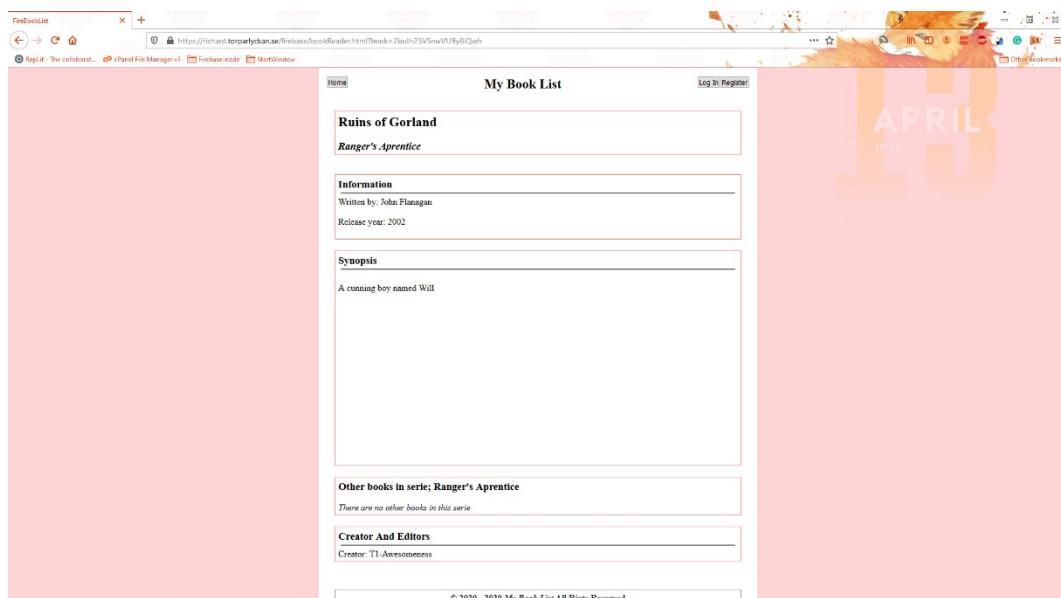
```

6.3 KLIENTVY

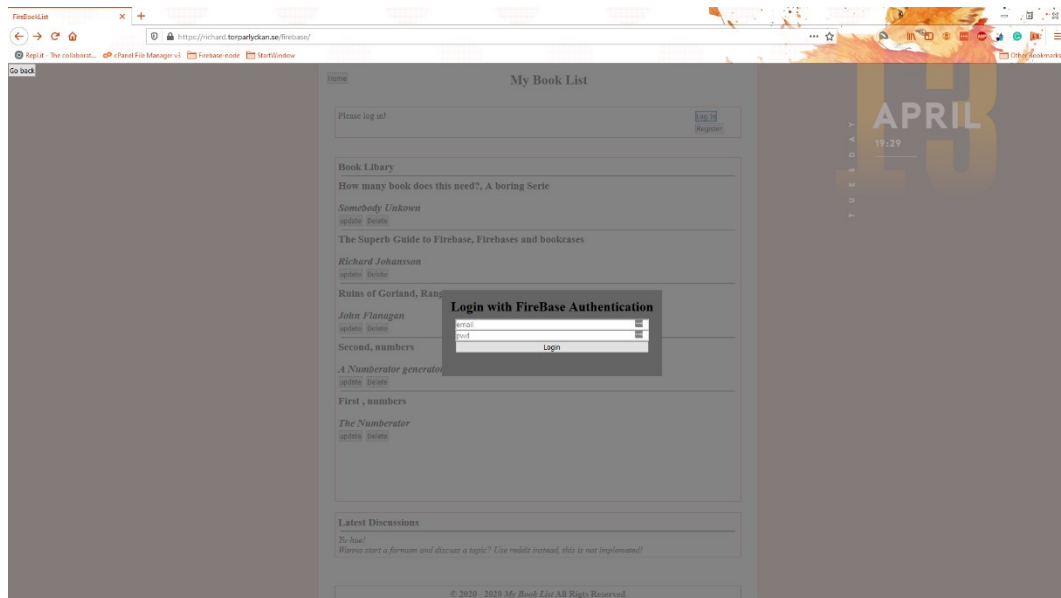
6.3.1 Startsidea



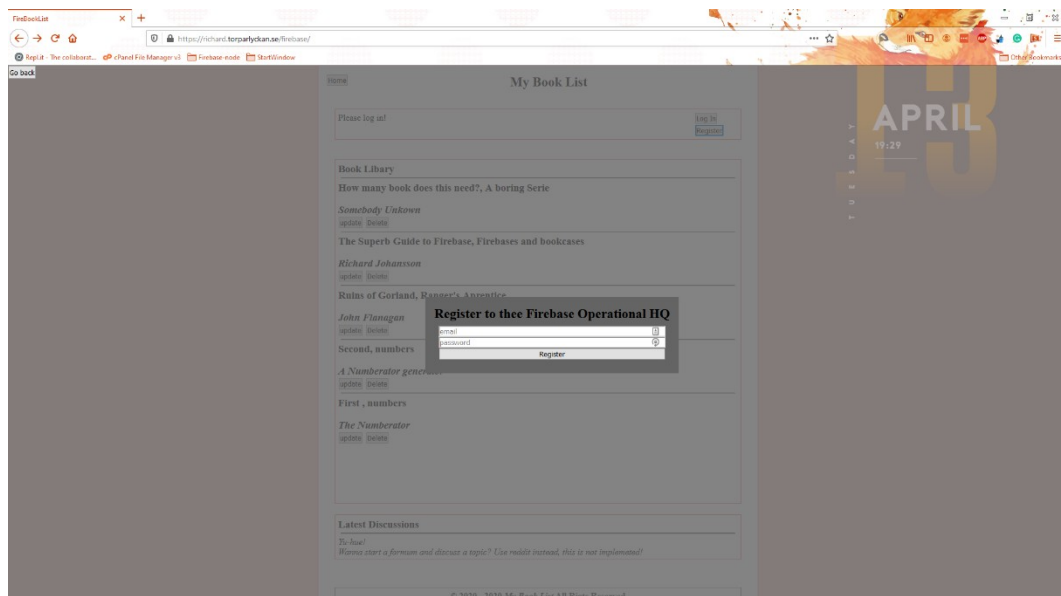
6.3.2 Bok-vy



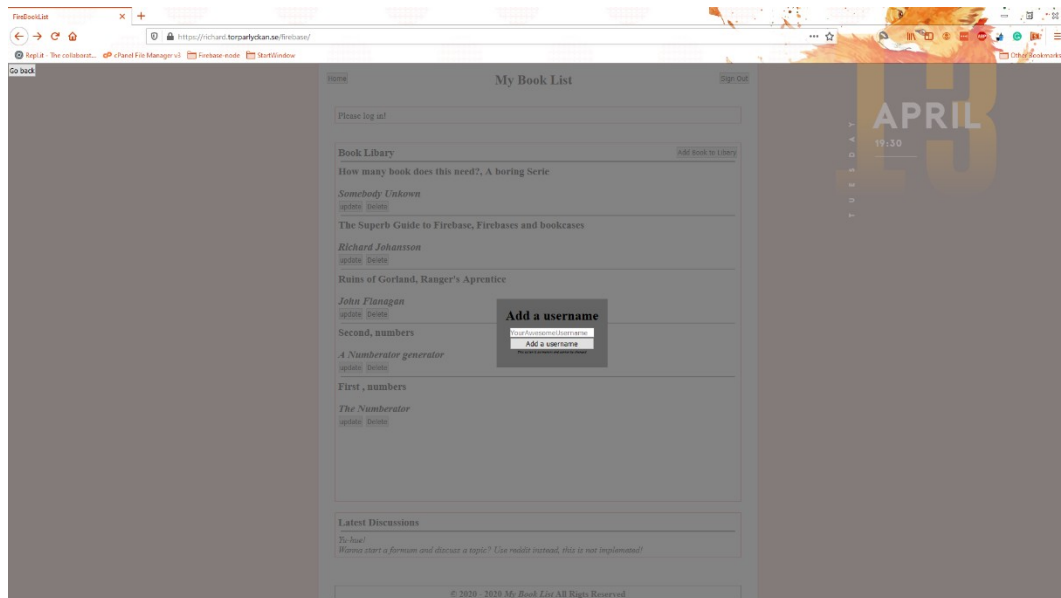
6.3.3 Login-vy



6.3.4 Registrerings-vy



6.3.5 'Add a username'-vy



6.3.6 Lägg till bok i bibliotek-vy

