Danmarks Tekniske Universitet

DTU

It og kommunikation
62581

Delaflevering 3

Gruppe 4

Andreas Bach Berg Nielsen s205869
Emilie Bracht Nielsen s205484
Julie Falsig Valeur s205485
Naveed Imam Shah s205491
Troels Engsted Kiib s205492
Jesper Clement Aaløse s205488

December 27, 2023 Github :

`https://github.com/Troels21/IT3-Delopgave-3`
Hjemmeside: `https://grp4.it3.diplomportal.dk`

Abstract

In this report the assignment of adding minimal security to our program was given. We accomplished this by implementing a variety of security solutions. Firstly we use the prepared statement object to protect the sql-database against sql injections. Secondly we hash and salt our passwords to achieve encryption as well as integrity. Thirdly we implement JavaWebTokens to remove validation from the frontend to the backend and hereby creating a reliable method to control login status. Lastly we use TLS to achieve confidentiality, data integrity, server authentication, and client authentication. Furthermore, we discussed the challenges that arose and the process of sharing and getting data from the other groups.

Indholdsfortegnelse

1 Introduktion

I denne rapport vil vi bygge videre på de to tidligere udførte delrapporter. Fokuset i denne rapport er primært at indføre minimal sikkerhed til vores tidligere udførte software. Vi har i denne opgave arbejdet på at kryptere vores brugeres kodeord. Til dette bruges hashing og salt. Yderligere til at validere vores brugerne, vil vi implementere authentication i form af tokens. Derudover vil vi også gerne kunne tilgå de andre gruppers data, og de skal kunne tilgå vores.

2 Kravspecifikation

Til denne rapport ønskes det at der indføres minimalt sikkerhed til programmet, samt en protokol for hvordan de forskellige grupper kommunikere med hinanden. Vi tolker kravet minimalt sikkerhed som, at vi ikke gemmer passwords i cleartext, samt salter vores password i databasen, en token validering og TLS. Ud fra det har vi kunne udarbejde disse krav:

- K1. Hashing og salting af passwords på SQL database
- K2. Gemme SQL database connection password på server niveau og ikke i kode.
- K3. TLS
- K4. Token validering
- K5. Protokol til hvorledes programmet skal interagerer med de andres programmer i XML-format.

3 Analyse

I dette afsnit vil vi gennemgå vores analyse af overførselsen af dataen fra andre grupper og en analyse af vores definition af ordet minimal sikkerhed.

3.1 Overførsel af data fra andre grupper

Da vi gerne vil have en dataoverførsel med de andres programmer, bliver vi nødt til at aftale protokollen af denne. Deraf bliver vi nødt til at holde et møde, hvori vi aftaler XSD-struktur, endpoint placering og hvad endpoint skal returnere.

3.2 Hvad vi mener med minimal sikkerhed

Da sikkerhed er et ikke-specifikt ord, og ordet minimal sikkerhed er endnu mere svagt begreb, er vi nødt til at definere, hvorfor vi mener at vores krav ville opfylde begrebet minimal sikkerhed.

3.2.1 Hashing og salting af passwords på SQL database

Hvis en kode gemmes på en database, uden nogen form for kryptering, kan man snildt få adgang til koden ved enten at hacke databasen eller injecte den med et rainbowtable-angreb. Dette betyder at der ikke er nogen sikkerhed i håndteringen af kodeord, når man gemmer koden direkte på databasen. Da man bare kan tilgå databasen igennem de førnævnte metoder, mister kodeordet sin reelle værdi, idet at meningen med et kodeord er, at andre ikke skal have kendskab eller adgang til denne. Derfor skal koden, inden den bliver lagt op på databasen, krypteres. Når man krypterer en kode, kan man nemlig ikke få kendskab til den oprindelige kode, men kun den krypterede kode, på databasen. Derudover er der også flere envejskrypteringer, så man aldrig

kan komme frem til den originale kode, ud fra det krypterede kodeord. Derfor ville kryptering beskytte mod injections, da man så kun ville kunne komme frem til den krypterede og aldrig den oprindelige. Hvis man kun krypterer på denne måde, kan man komme i en situation hvor to personer har samme oprindelige kode og derfor også samme krypterede kode på databasen. Dette mindsker sikkerheden, da man kan regne ud hvilken oprindelige kode andre har, med rainbow inection, hvis man opdager at ens egen krypterede kode er identisk med en andens krypterede kode. Derfor skal der ændres i koden, med noget som gør dét kodeord til unikt, inden det krypteres. Koden får tildelt et unikt "salt" som et præfiks på koden, som er en værdi, der vil være forskellig fra person til person, når det bliver krypteret. Personen selv har ikke kendskab til denne værdi. Derfor skal man, når man gemmer en kode i en database, også salte den oprindelige kode, altså gøre det unikt, og derefter hashe, altså lave den egentlige kryptering af det oprindelige kode. Dette fører til en sikre databasen, hvor enhver kode er krypteret og unik, så hvis den bliver angrebet udefra, vil man hverken kunne komme frem til den originale kode eller kunne se om flere koder er ens. Derfor skal man, når man prøver at brute force sig frem, gøre dette for hver enkelte, altså først komme frem til saltet, og derefter bruteforce hashet. Dette tager statistisk rigtig lang tid at gøre, da man ved at injecte en rainbow tabel skal igennem 490GB data for hver kode, også er det ikke engang sikkert, man kommer frem til koden. Derfor vil man, ved at hashe og salte koden, skabe et højt niveau af sikkerhed på de koder som der gemmes på databasen.

3.2.2 Gemme SQL database connection password på server niveau og ikke i kode

Da et password ikke må stå i cleartext, skal vi få denne gemt på et serverbaseret niveau. Dette vil vi gøre fordi vores server allerede er beskyttet via. SSH protokollen og derfor ville vi mene at passwordet til databasen er gemt nok.

3.2.3 TLS

Fordi vi overfører cpr-nummer samt notater til cpr-nummeret, vil vi mene at en del af dataen gerne skulle føres igennem nettet krypteret, da det er sårbare data. Derfor vil vi mene, at man ikke kan snakke om minimal sikkerhed i vores program, uden vi får inkorporeret TLS eller SSl, til netop at skabe "confidentiality, data integrity, server authentication, and client authentication".

3.2.4 Token validering

Da vi skal sørge for, at det kun er de rigtige brugere, der har mulighed for at se og interagere med vores hjemmeside, har vi behov for en login validering. Denne validering vil vi gerne have til at fungere med en token, således at vi senere har mulighed for at se, om det stadig er den samme bruger der interagerer med vores hjemmeside.

3.2.5 Vores samlede sikkerhed

Alt dette vil give os mulighed for at folk ikke ville kunne opsnappe data, de ikke skulle have haft, at det kun er de rigtige brugere der kommer ind i vores program og at det er svært at direkte gætte vores kodeord. Alt dette til sammen vil vi mene skaber en rundhåndet minimal sikkerhed i vores program.

4 Design

I dette afsnit vil gennemgå de ændringer, der er i vores design klasse diagram og ER-diagram og sekvensdiagrammer. Derudover vil vi også gennemgå hvordan protokollen for overførslen af de forskellige gruppers data.

4.1 Overførsel protokol

Vores overførsels protokol blev defineret ved følgende parametre:

- Alle opgiver en sti til deres endpoint der slutter således: /aftaler?cpr=xxxxxxxxx

  Hvis man tilføjer et cpr-nummer, skal man få alle aftaler til det pågældende cpr-nummer.

  hvis man ikke tilføjer et cpr-nummer, skal man få alle aftaler i databasen.

- Man skal have en aftale liste der indeholder aftaler.

- En aftale indeholder:

  CPR: $string(<< required >>)$

  ID: $String(<< required >>)$

  klinikID: $String(<< required >>)$

  timeStart: $YYYY - MM - DD\ HH : MM.(<< required >>)$

  timeEnd: $YYYY - MM - DD\ HH : MM.(<< required >>)$

  notat: $String(255)(<< optional >>)$

- Alle skal skrive deres endpoint ind i et delt dokument.

Ud fra disse parametre har vi lavet et XSD dokument, der giver et overblik, og giver os mulighed for, at autogenere klasserne ved hjælp af codehaus jaxb2:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="aftaleListe">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="aftale">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CPR" type="xs:string" />
              <xs:element name="ID" type="xs:string" />
              <xs:element name="klinikID" type="xs:string" />
              <xs:element name="notat" type="xs:string" />
              <xs:element name="timeEnd" type="xs:string" />
              <xs:element name="timeStart" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 1: XSD

Ud fra denne protokol, kan vi designe hvordan dette skal implementeres i koden. Deraf har vi lavet et sekvens diagram, der beskriver flowet for hvordan vi bearbejder dataen, hvis vi gerne vil vise den på vores frontend.

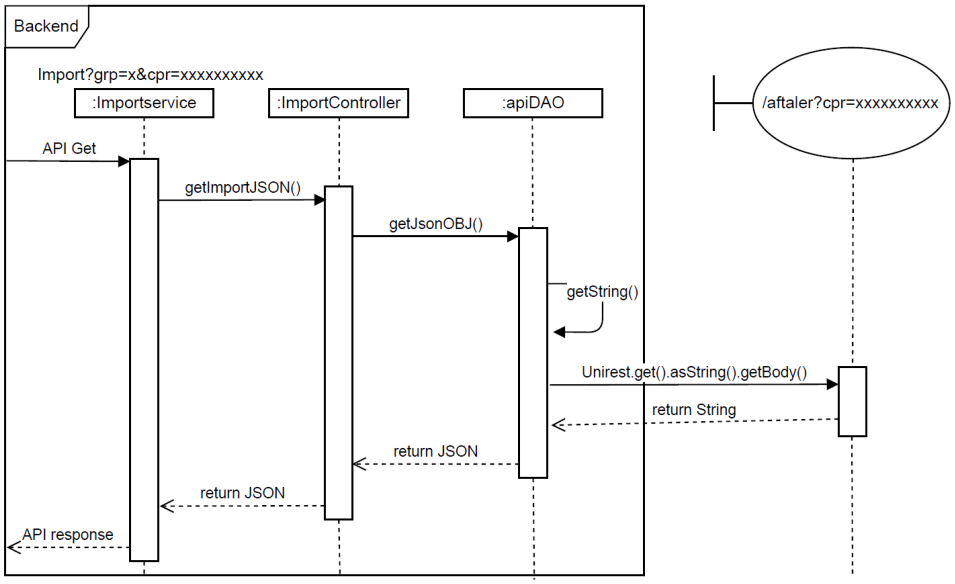

Figure 2: Sekvensdiagram over overførsels protokollen

Vi vælger ikke at designe/implementere en måde således, at vi kunne gemme de andres gruppers data, da vi må vurdere, at flertallet af grupperne ikke har implementeret den aftalte protokol. Derfor har vi kun designet en måde at vise dataen i frontenden, således at vi kan vise, at kravet K5. bliver opfyldt til en vis grad. Vi vil have, at vores program skal kunne kalde vores eget endpoint og bearbejde data'en, som var den en af de andre gruppers data. Derfor viser vi kun dataen, istedet for at overfører dataen til vores egen database/aftalesystem.

4.2 Hashing og salting af passwords

Vi har tænkt os at lave et hash og salt på vores password til vores bruger. Dette skal gøres således, at vi først salter brugerens kodeord med en random værdi, hvor vi derefter hasher det. Bagefter skal vi uploade den hashede kode og det unikke salt til databasen. På denne måde vil vi kunne hente saltet ned, sætte det på den kode brugeren skriver ned, hashe det, og derefter sammenligne det med den hashede kode på databasen. Dette er der lavet et ER-diagram over, hvor vi kan se hvordan vi har tænkt os at gemme det i databasen, under tabellen Loginoplysninger.

4.2.1 ER diagram

ER diagrammet viser en oversigt over hvilke tables, med tilhørende entities, cardinality og attributter, der er oprettet i SQL-databasen

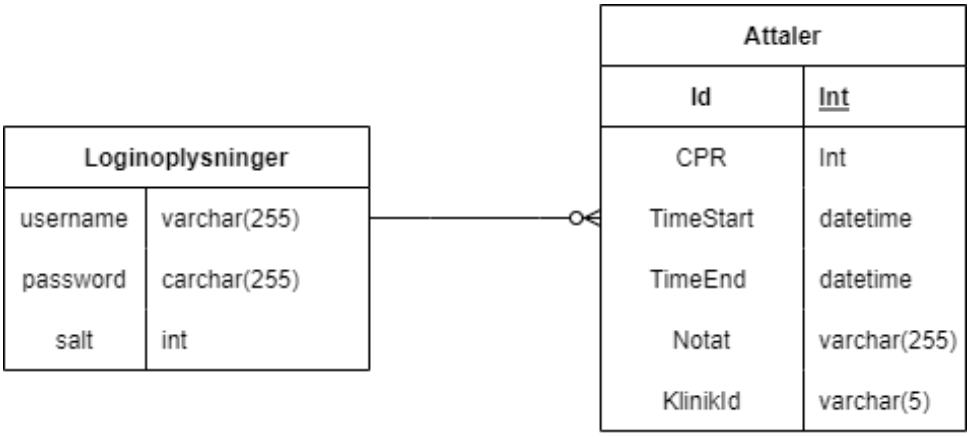

Figure 3: ER diagram

4.3 Logind validering.

Vi har udarbejdet et sekvensdiagram der viser flowet i vores login validerings sektion.

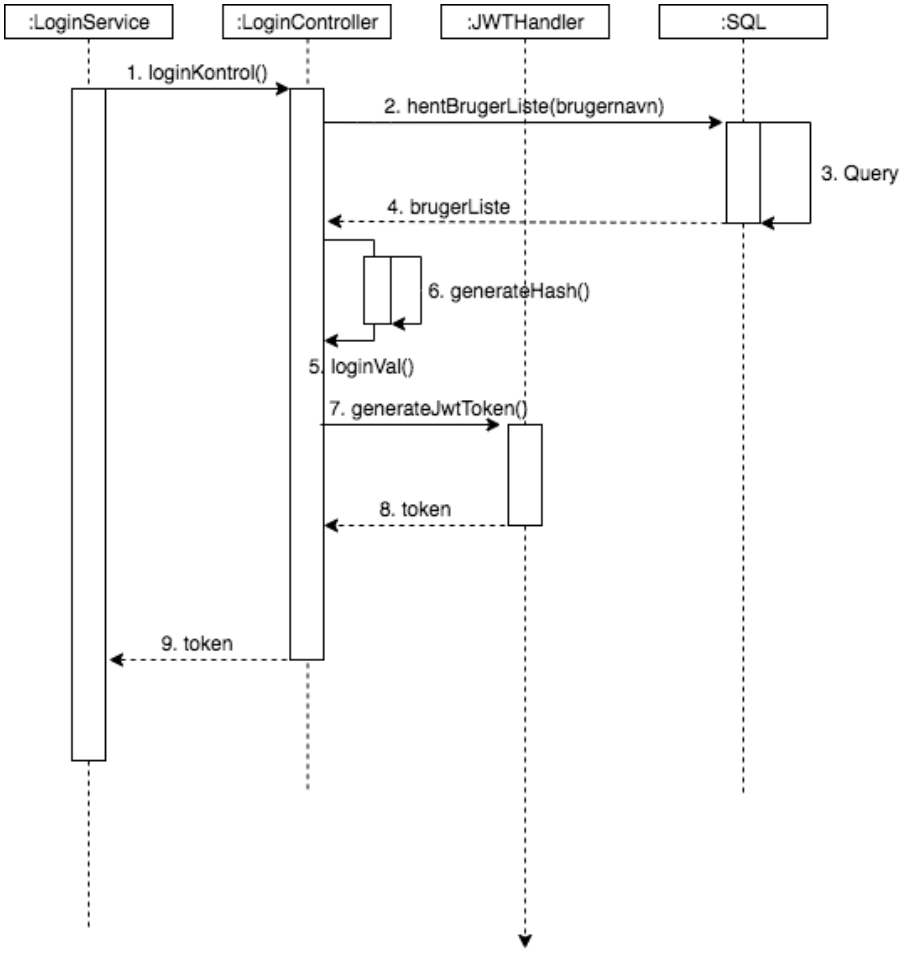

Figure 4: Sekvensdiagram

4.4 Design klasse diagram

For at skabe det størst mulige overblik, har vi valgt at dele design klasse diagrammet op i 3 dele:

- Log ind
- Aftaler
- Import

4.4.1 Login

På dette diagram viser et designklassediagram over alle klasserne der har log ind processen at gøre. Dette inkluderer altså både frontend, hvor brugernavn og kodeord indtastes, backend hvor brugerens data bliver bearbejdet og databasen som har brugerlisten.

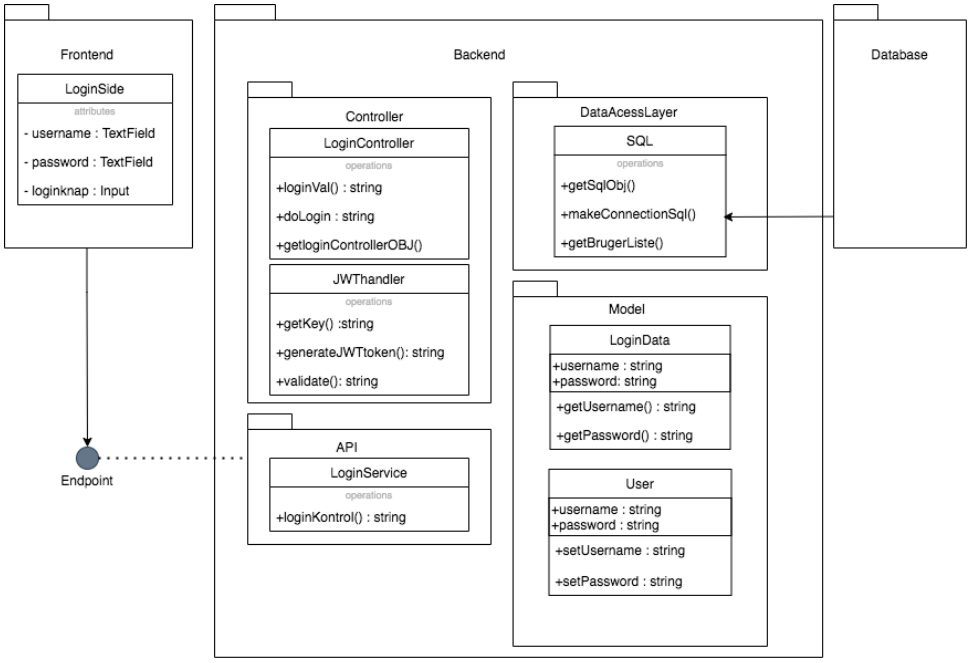

Figure 5: Design klasse diagram - Log ind

4.4.2 Aftaler

På dette diagram ses et design klasse diagram over alle klasser, der har med oprettelse og fremvisning af aftaler. Her ses hvordan de forskellige klasser skal fordeles over frontend, hvor man kan oprette en ny aftale, og se nye aftaler i kalenderen, backend hvor aftalerne bliver bearbejdet og databasen hvor alle aftalerne bliver gemt.

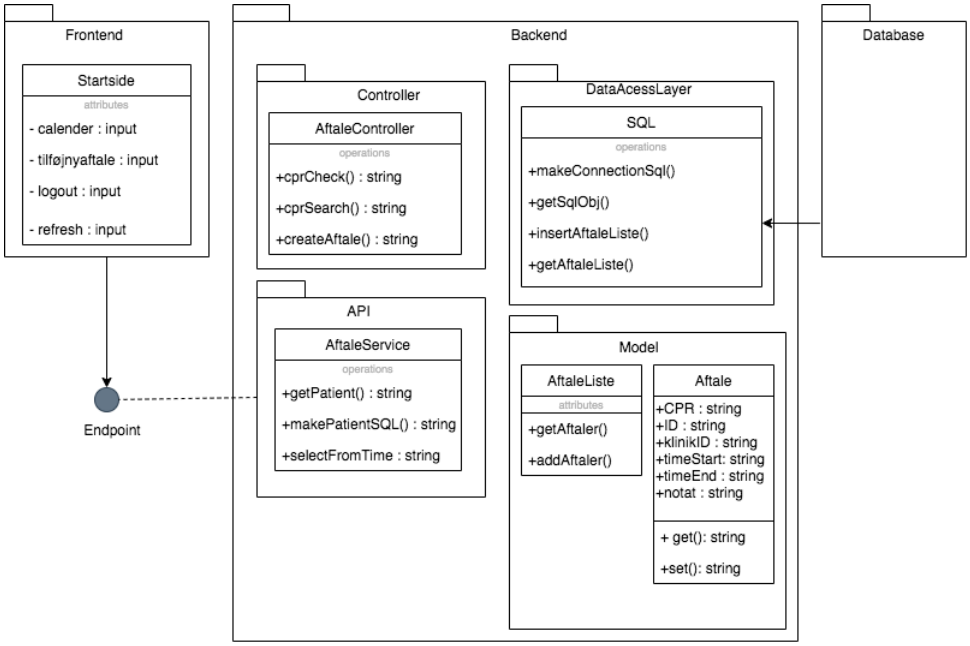

Figure 6: Design klasse diagram - Aftaler

4.4.3 Import

På dette diagram visses et design klasse diagram over import processen. Her ses hvordan de forskellige klasser fordeles over frontend, hvor man visuelt kan se aftalerne der er importeret og backenden. Herefter vil vi omskrive det til Json, hvor de forskellige aftaler skal blive hentet af de forskellige grupper.

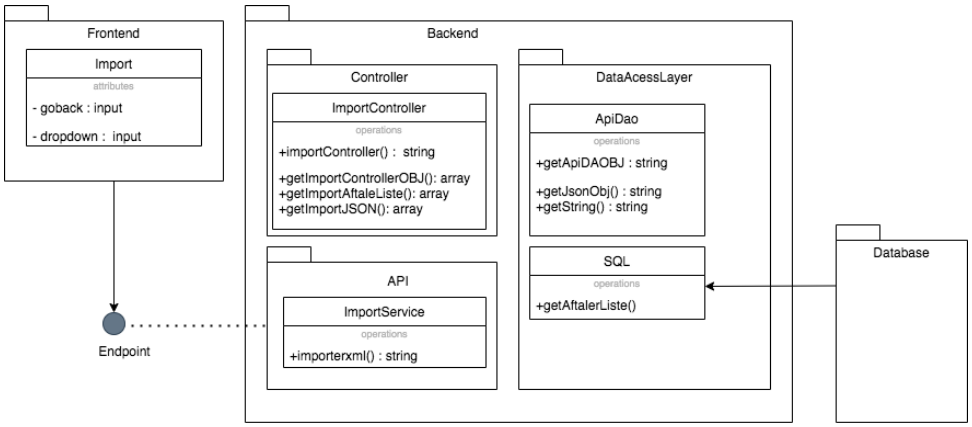

Figure 7: Design klasse diagram - Import

5 Implementation

I dette afsnit vil vi gennemgå hvordan vi har implementeret:

- SQL connection password
- Hashing og salting
- Token

- TLS
- Overførsel af data fra andres grupper

5.1 SQL connection password

Vi havde før SQL-connection passwordet stående i cleartext i vores backend, dette password er nu blevet inkorporeret som et system environment variable istedet. Dette system environment variable er blevet sat ind i ubuntu som en permanent environment variable, da adgang til vores ubuntu foregår via. SSH protokollen.

5.2 Hashing og salting

Vi starter ud med at lave en getSalt() metode, som skal lave en array med random numre, og derefter tager den et af de numre og returnerer. Denne metode bliver ikke brugt, da vi hverken laver eller ændre brugerens kodeord. Derefter hasher vi ved brug af MD5 som er en message-digest algorithm, der bruges til at hashe en værdi. Denne metode hedder generateHash(String pass, int salt). I denne metoder bruger vi to argumenter. Pass kommer fra loginsiden og er den værdi, som brugeren skriver som kode. Salt kommer derimod fra databasen, hvor vi fetcher saltet tilhørende det brugernavn der bliver skrevet. Derefter lægger vi de to sammen og laver en return på en 32 bit lang hash. Hvis det ikke kan lade sig gøre, returnerer vi pass.

5.3 Token

Vi har implementeret tokens ved først at sende loginoplysninger til backenden og derefter ved at validere dem. Efter loginoplysningerne er valideret, bruges de til at konstruere en token. Vi bruger specifikt et user-objektet og en key dannet fra signaturalgorithmen HS512 til at lave vores token. Der oprettes en ny key ved hver session og denne key opbevares på serveren.
Den nye token returneres så i form af en streng til frontenden. Modtagelsen af en token gælder som valideringen og brugeren sendes derfor videre til startsiden. Når andre funktioner bruger et fetch-kald, får de så en token med i deres header. Filteret fanger fetch-kaldene inden de udføres og kontrollere om de har en valid token ved brug af token-handler-validerings metoden. Hvis der ikke er en valid token smides en 401 authentication error. Vi har valgt ikke at lave token validering af vores endpoint "aftaler" fordi det skal være muligt for de andre grupper at hente data fra dette endpoint uden en login token.
Til sidst er der if-statements. Her bruges tokens til skabe useability ved at føre brugeren til loginsiden hvis de mangler en logintoken.

5.4 TLS

Vi har implementeret TLS på vores tomcat-server via. Letsencrypt. Til dette har vi installeret nginx, snapd og certbot på ubuntu vm-maskinen. Vi fik et certifikat igennem certbot og nginx, og prøvede at proxypass nginx vidre til vores Tomcat server. Dette arbejdede vi med i nogle timer uden held, så derfor brugte vi den nemme løsning og kopierede vores certifikat filer ind i Tomcat manuelt. Dette medfører nu, at når cerfikatet udløber bliver man manuelt nødt til at gå ind og kopiere det nye certifikat, der er sat til automatisk fornyelse i nginx, over i Tomcat igen. Dette er ikke særlig smart lavet, men som beskrevet havde vi problemer med, at lave den gode løsning, som havde været et proxypass.

5.5 Overførsel af data fra andres grupper

Først har vi lavet et data-access-objekt, hvor vi har mulighed for at hente de andres gruppers aftaleliste med aftaler i XML-format og læser disse lister som en string. Dernæst har vi lavet en metode der konvertere denne string til et JSON objekt, da vi vil mener at det er meget nemmere at arbejde med JSON end XML. Disse metoder bruger vi i vores ImportController, hvor vi har to metoder. Den ene returnerer bare JSON-objektet videre til ImportService og den anden laver JSON objektet om til et Aftaleliste-objekt. Dette gør vi fordi vi gerne på et senere tidspunkt vil hente de andres gruppers aftale-data og inkorporere det i vores data. Vi vil derfor gerne have alt dataen i en stor aftaleliste som vi kan vise. Vi har ikke kunne implementere disse funktioner ordentligt endnu, da de andre grupper ikke er klar til dette step endnu. Til slut har vi et ImportService endpoint der indeholder en switch, der differentierer imellem de forskellige grupper og cpr. Denne service kalder de andre klasser og får derved et JSON-objekt til det ønskede data og returnerer dette via. et JSON objekt.

Det åbne endpoint, de andre grupper kan bruge til at tilgå vores aftaleliste, bliver kontrolleret af vores filter, der kontrollerer hvorvidt man har det korrekte authorization header med.

6 Afprøvning

I dette afsnit vil vi afprøve K5. Da vi mener, at dette krav er det eneste, som ikke allerede er dokumenteret direkte i koden eller et offentligt certifikat.

6.1 K5.

Vi vil afprøve 5 forskellige scenarier:

- A1. Vi vil afprøve om vores endpoint kan returnere hele vores aftaleliste, når man har den hemmelige nøgle.
- A2. Vi vil afprøve om vores endpoint kan returnere enkelte cpr fra vores aftaleliste, når man har den hemmelige nøgle.
- A3. Vi vil afprøve om vores endpoint returnerer noget, hvis cpr ikke eksisterer, når man har den hemmelige nøgle.
- A4. Vi vil afprøve om vores endpoint kan returnere noget, når man har en forkert hemmelige nøgle.
- A5. Vi vil afprøve om vores endpoint kan returnere noget, når man ikke har en hemmelig nøgle.

6.1.1 A1

Som det kan ses i bilag A18 lykkedes det at returnere aftalelisten ved hjælp af postman applikationen. I dette scenarie satte vi den private nøgle til: "hemmeliglogin" i vores backend. Der returnerer succesfuldt en streng, der indeholder vores aftaleliste i XML format.

6.1.2 A2

Vi giver API kaldet parameteren cpr=1234567893, som vi ved eksisterer i databasesen, og har en enkelt aftale. Som det kan ses i bilag A29, returnerer serveren succesfuldt en aftale til det givne cpr.

6.1.3 A3

Vi giver API kaldet cpr=1234, som ikke eksisterer som CPR i databasen og det korrekte hemmelige kodeord "hemmeliglogin". Vi kan se på kroppen af beskeden, at der returnerer en tom aftaleliste.10

6.1.4 A4

Som tidligere nævnt er den hemmelige nøgle sat til at være "hemmeliglogin" i backenden, og derfor når vi sætter Authorization til at være "forkerthemmeliglogin" returnerer den status 401, den smider altså den korrekte exception og dataen returneres ikke.11

6.1.5 A5

Hvis der slet ikke gives en token med, kan programmet godt håndtere dette uden at lave en nullpointer exception, og smider korrekt en 401 exception.12

7 Diskussion

I dette afsnit vil vi gennemgå fejl/udfordringer samt andre elementer, vi mener der skal diskuteres. Dette er blandt andet hvilke problemer vi er stødt på, men til dels er det hvilke forbedringer eller ændringer vi forventer, vi gerne vil lave i vores program senere.

7.1 Resultat af afprøvning

Vi fik succesfuldt gennemført fem cases hvor vi afprøvede det femte krav. Afprøvningen af de andre krav kan ikke lade sig gøre eller ville være for omstændeligt iht. dette projekt. Det er også problematisk at vi selv udføre afprøvningen af vores sikkerhed, da vi kun tester det, vi ved er sikret. En eventuelt forbedring til dette projekt vil derfor være, at få en anden gruppe til at prøve at tilgå databasen eller få en token uden brug af korrekt login og dokumentere deres forsøg. Det er også sandsynligt at der eksistere huller i vores sikkerhed, som vi slet ikke kender til, og derfor ville det være gavnligt at få en ekstern person med yderlige kompetance til at teste sikkerheden.

7.2 Fejl/udfordringer

Vi har stadig fejl, mangler og udfordringer i vores program.

7.2.1 Import

Da vi importerede data fra aftale-endpointet kom vi frem til, at hvis vi gerne ville have en aftale frem eller tilgå et cpr som kun havde en enkel aftale, ville den ikke printe det ud. Dette gav for os ingen mening, da vi kunne se at programmet hentede det fint og at vi også kunne printe det ud ved brug af console.log(). Her prøvede vi med try/catch, da vi troede at den måske ville lave en fejl når den ikke kunne printe. Dette var dog ikke tilfældet, da vi fandt ud af ved brug af print statements, at den slet ikke kom ind i catch. Derfor endte vi med at opdele det i to metoder. Vi har en metode hvis der er flere aftaler, så bruger vi createList() og en metode hvis der kun er én aftale bruger vi createSingle(). På denne måde slipper vi for at bruge en løkke i den metode som alligevel kun har en enkel aftale. Dette gjorde at vi så kunne vise en aftale og dermed løste problemet.

7.2.2 Endpoint til de andre grupper

Som koden fungerer lige nu skal man bare overføre et cleartext hemmeligkodeord med. Dette et på ingen måde nok, men planen er at aftale en syncron hemmelig nøgle med de andre grupper, som er saltet og hashet istedet. Dette er dog ikke blevet udført endnu, men koden er på plads.

7.3 Hash og salt af nye brugere

Hvis vi skulle implementere at oprette nye brugere, så ville vi kunne benytte metoderne getSalt() og generateHash() til at kryptere deres kode. Derefter skulle vi lave en SQL metode som executer en query der uploader denne værdi til databasen. Derfor kan vores metoder godt fungere. Vi har også brugt metoderne seperat til at danne det hash og det salt der bliver brugt til den enkelte bruger vi har på databasen. Derfor vil denne implementering ikke være et problem, men mere et spørgsmål om at lave en SQL metode, der inserter værdierne i tabellen Loginoplysninger.

7.4 Fremtidige forbedringer

Vi vil mene at programmet aldrig er fejlfrit og på baggrund af denne holdning, har vi konstrueret nogle eventuelle forbedringer.Der er utallige ting, vi kunne tage fat på til fremtidige forbedringer. Derfor vælger vi kun at drøfte de mest væsentlige og dem der er aktuelle.

7.4.1 Salting og hashing af syncron hemmelig nøgle

Vi skal have fået lavet en hemmelig kode og overført denne på en sikker måde til de andre grupper. Derudover skal denne saltes og hashes på en måde, der giver denne kode større sikkerhed.

7.4.2 Flere brugere

Der skal kunne oprettes nye brugere eller ændre deres nuværende kodeord. Dette er ikke en del af minimalkravene, så derfor er det ikke implementeret. Ellers ville man kunne gøre det ved at have en opret bruger knap der saltede og hashede kodeordet og derefter med en preparestatement, der uploader værdierne til databasen. Dette vil så forudsige at vi så kan skrive eller notere brugeren, der laver aftalen. Hvilket vil gøre, at hvis en anden bruger kigger vil vi kunne finde hvem det er der har oprettet det, for at afklare mulige spørgsmål mm. Udover dette vil vi også kunne differentiere mellem ledige tider, f.eks hvis en bruger har en aftale kl. 7 men en anden ikke har, så vil man stadig kunne lave en aftale kl. 7, bare med en anden bruger.

7.4.3 Patient tabel

En anden mulig forbedring, som ligger sig op ad flere brugere er, at man kan oprette patienter og gemme deres data med deres CPR. Dette vil gøre at man så kun skal skrive CPR for at få alle oplysningerne til at lave en aftale. Dette gør også at man kan tilkolbe alle personens aftaler sammen til det CPR, så man kan søge på en specifik patient og kun få dens aftaler. Det ligger sig tæt op af import måden at filtrere med CPR, men det er fundamentalt anderledes, da det ikke vil blive brugt i import, men nærmere i startsiden.

7.4.4 Kun tilgængelige tider

Når man opretter en aftale, vises alle tidspunkter på dagen og brugeren kan derefter vælge det ønskede kvarter, hvilket er ikke er specielt effektivt og kan være lidt uhensigtsmæssigt. Derfor vil en mulig forbedring være at vise tiderne i kvartaler, og kun de tider indenfor tidsrammen. Derudover vil det være en forbedring, kun at vise mulige tidspunkter. Så hvis kl.7 er taget,

så er det ikke en mulighed. Hvis vi implementerer dette hvor vi har flere brugere, skal vi også kunne vælge hvilken bruger man gerne vil oprette en aftale med og derfor vise aktuelle tider for alle og kun hvis ingen bruger har en ledig tid kl.7 vil den ikke vises.

7.4.5 CSS

Vi har opdaget at vores html elementer godt kunne kræve lidt bedre CSS. Dette er fordi at på forskellige størrelse skærme, eller hvis man zoomer ind og ud, vil nogle elementer ændre sig markant så de ikke er på deres tiltænkte pladser.

7.4.6 Overensstemmelse af sprog i koden

I vores kode har vi skrevet nogle ting på dansk og nogle ting på engelsk, dette giver først og fremmest et dårligt overblik over koden, da der ikke er konsekvens brug af samme sprog. Udover dette subjektive problem kan dansk sprog i kode give problem under compilingen af koden, som jo er et mere objektivt problem. Derfor vil vi gerne omskrive alt det danske i vores kode til engelsk, men vi må bare acceptere, at denne opgave ikke har været prioriteret, da fokus har været på en række andre opgaver.

8 Konklusion

Tager man et kig på vores kravspecifikationer:

K1. Vi gemmer nu vores password i LoginOplysninger tablen i databasen, som en saltet og hashet MD5 værdi.

K2 Vi gemmer nu vores password til SQL-connenction via. system environment variabler i stedet for cleartext.

K3. Vi har fået et TLS certifikat fra LetsEncrypt og implementeret dette i vores tomcat-server. Vi har dog ikke opnået at proxypass igennem nginx, dette betyder at når certifikatet løber ud, skal det manuelt overføres fra nginx til tomcat.

K4. Vi har implementeret en token-valididering med et filter, der kontrollerer hvorvidt man har en token og om man har det rigtige token.

K5. Vi kan interagere med de andres programmer i XML-format. Deres programmer har dog fulgt den bestemte aftale protokol endnu, samt at dette krav for rapportten er blevet rykket ind i semesterprojektet. Derfor har vi implementeret denne funktion, til at fungere på vores egen API. Det giver os mulighed for, at vise hvordan vi regner med at kommunikere med de andre gruppers programmer, når de når så langt. Dette er dokumenteret i afprøvningen af kravet K5 og deraf kan vi konkludere at dette krav er opfyldt.

På baggrund at disse vurderinger vil vi konkludere, at vi har lavet et program der opfylder vores kravspecifikationer. Der er dog stadig med mulige forbedringer, som er diskuteret i diskussions afsnittet.

9 Bilag

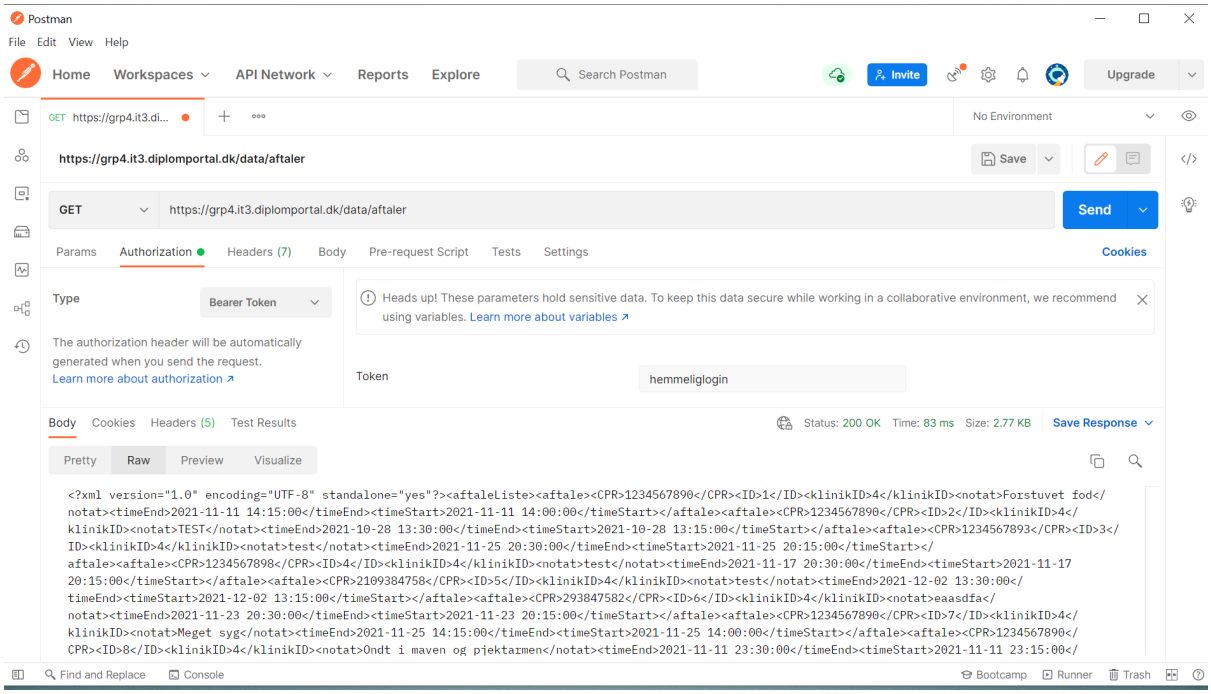

Figure 8: A1

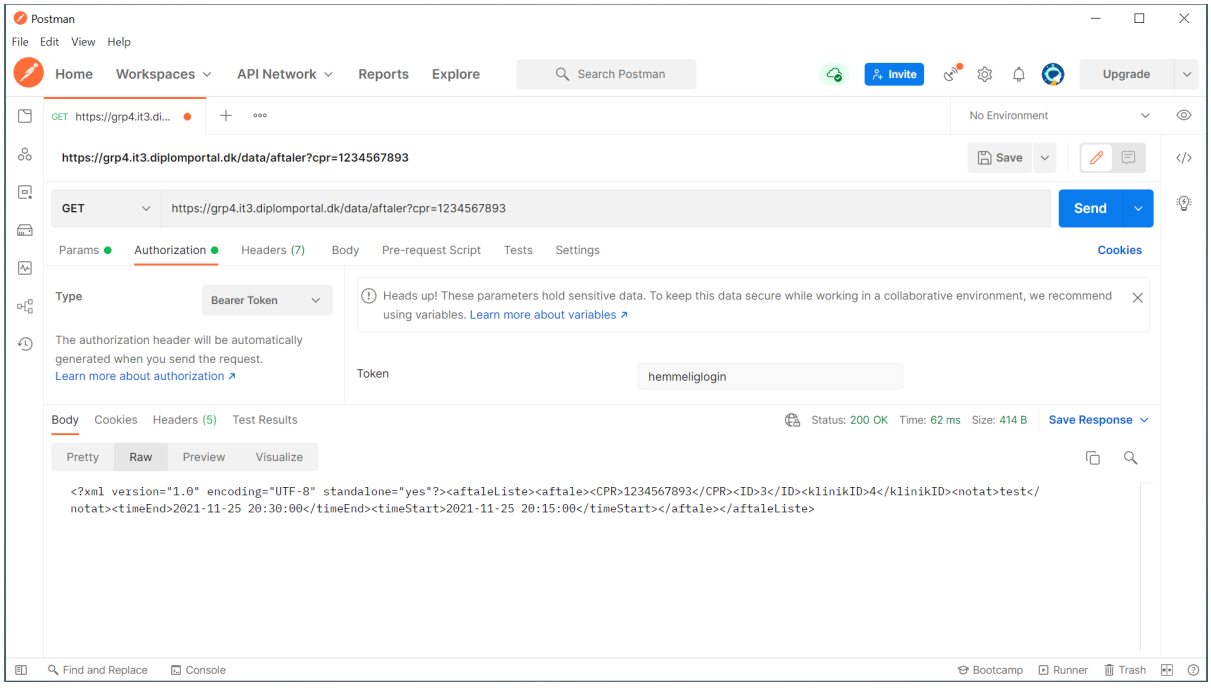

Figure 9: A2

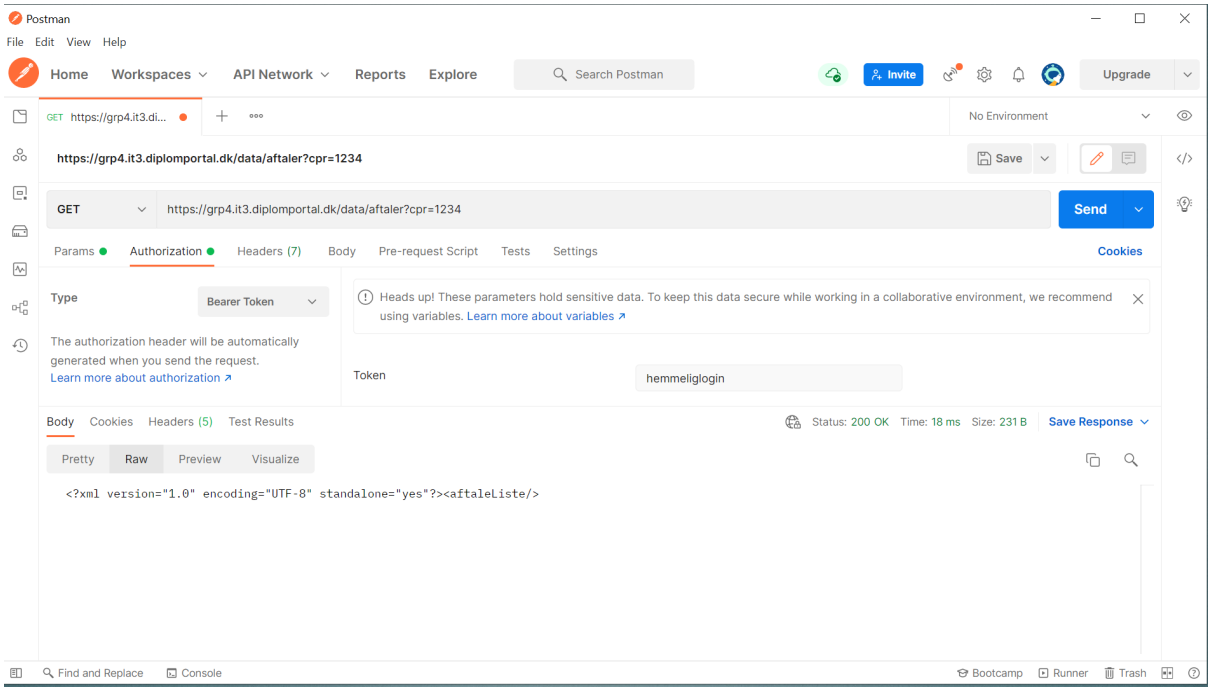

Figure 10: A3

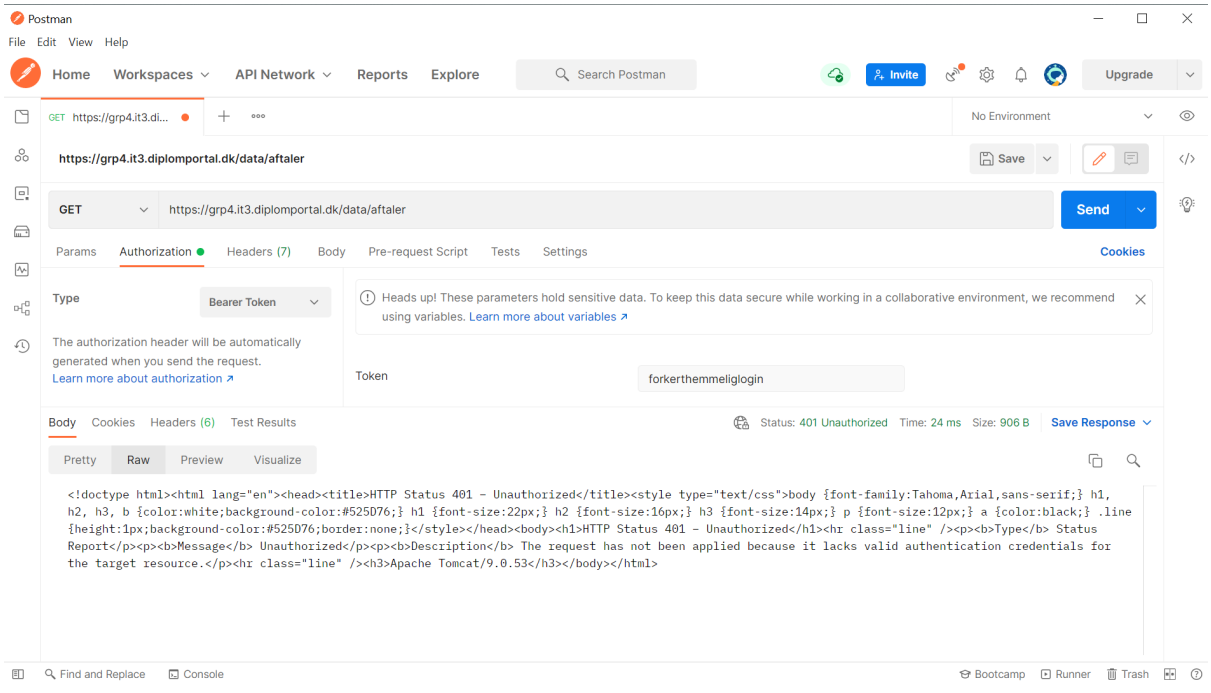

Figure 11: A4

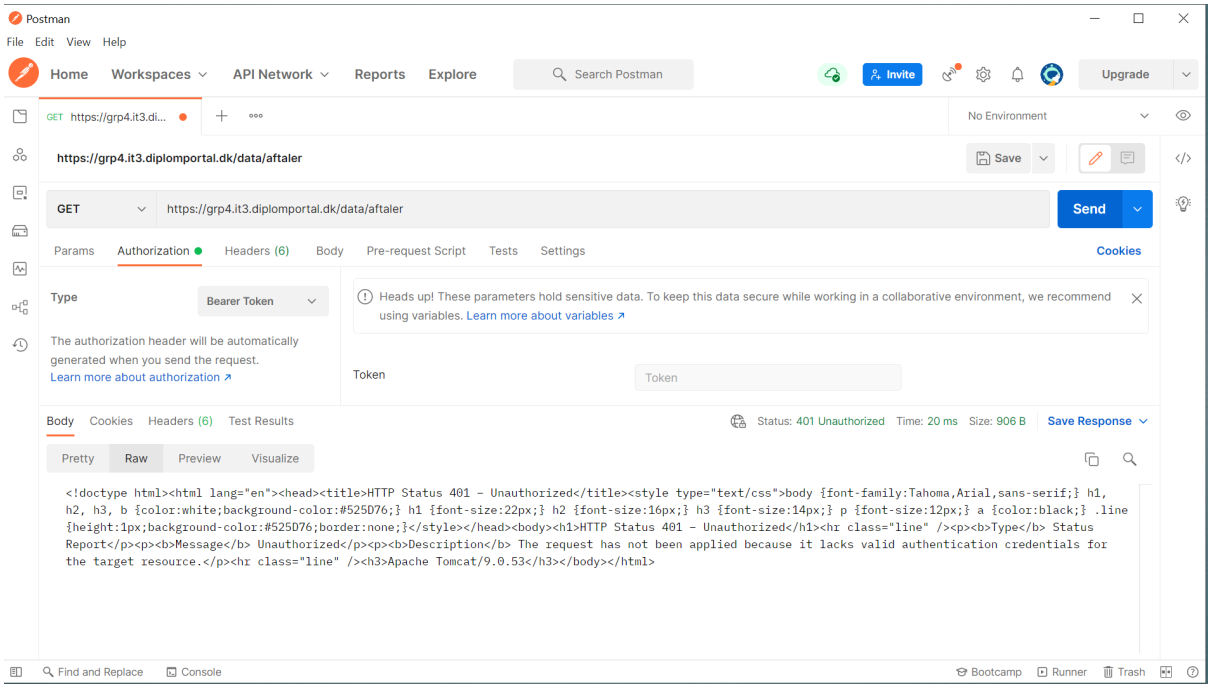
Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
GET https://grp4.it3.di...
No Environment
https://grp4.it3.diplomportal.dk/data/aftaler
Save
GET
https://grp4.it3.diplomportal.dk/data/aftaler
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
Type
Bearer Token
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables
The authorization header will be automatically generated when you send the request.
Learn more about authorization
Token
Token
Body Cookies Headers (6) Test Results
Status: 401 Unauthorized Time: 20 ms Size: 906 B
Save Response
Pretty Raw Preview Visualize
<!doctype html><html lang="en"><head><title>HTTP Status 401 – Unauthorized</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 401 – Unauthorized</h1><hr class="line" /> Type Status Report Message Unauthorized Description The request has not been applied because it lacks valid authentication credentials for the target resource. <hr class="line" /><h3>Apache Tomcat/9.0.53</h3></body></html>
Find and Replace
Console
Bootcamp
Runner
Trash

Figure 12: A5