Övningar:

# Cookies och Storage

**1.1** Skriv ett script som lagrar ditt namn och hemstad i localStorage.

**1.2a** Bygg en hemsida som visar hur många gånger användaren har besökt din hemsida.

**1.2b** Lägg till ett textformulär där en användare kan skriva in något. Nedanför ska det finnas en rubrik “Past Entries” som visar vad användaren har skrivit in i fältet förut.

**1.2c** Se till att “Past Entries” inte försvinner om man laddar om sidan.

**1.2d** Lägg till en timestamp på varje entry.

**1.2e** Lägg till en knapp som nollställer besöksantalet.

**1.3a** Gör en hemsida som innehåller en knapp. Om man trycker på knappen ska webbläsaren lagra en cookie med namnet clicks. clicks ökar med 1 för varje gång användaren trycker på knappen.

**1.3b** Lägg till en header på sidan. På denna ska det stå “Välkommen tillbaka, du var här för X sekunder sedan!”

**1.3c** Lägg till en knapp “Remove all cookies” som tar bort samtliga cookies på din sida.

**1.4** Bygg en hemsida som skriver ut alla cookies och deras värden i en tabell med två kolumner. Exempelvis:

| Namn | Värde |
| --- | --- |
| lastVisited | 2022-01-01T23:00:00.000Z |
| favoriteTheme | darkmode |
| itemsInCart | 2 |

# Sessioner

**2** Du ska i den här övningen bygga en sida för en onlinebokhandel. Nedan följer några features du kan implementera i valfri ordning. Du behöver inte implementera något betalsystem.

* Användare kan se olika böcker på startsidan som finns i bokhandeln. *Börja med denna.*
* När man klickar på en bok kommer man till lite information om denna. Du kan börja med bara några böcker och hårdkodade värden. Om du vill kan du lägga dessa i en databas. *Vänta med att lägga in värdena i en databas. Börja med hårdkodade värden.*
* En användare får på startsidan upp “Recently Viewed” med en lista på böcker den nyss kollat på. *Använd express-session och lägg recently viewed i en array.*
* En användare kan lägga till böcker i varukorgen. Vilka böcker som finns i varukorgen sparas i sessionen. *Använd även här express-session*
* En användare har ett inlogg där sessionsdata sparas mer permanent. Användaren kan nu stänga ner webbläsaren, komma tillbaka och fortfarande ha kvar varorna i korgen. *Först här behöver du redis.*
* Efter en viss tid av inaktivitet förstörs sessionen och användaren loggas ut. *Du måste lägga till en expires på kakan på din session, och kontinuerligt lägga på tid varje gång du skickar en request till servern.*

*Tips*

1. *Börja med att göra en index.html med hårdkodade böcker. Se sedan till att skicka den från en expresserver.*
2. *Skapa en endpoint som returnerar böckerna*
3. *Hämta böckerna via index.html (du kan importera ett skript här)*
4. *Skriv ut böckerna via en .forEach()*
5. *Lägg till knappar under varje bok i din .forEach() som leder till en POST /books*

Refaktorera gärna koden allt eftersom du lär dig bättre metoder att lösa problemen under kursens gång. Exempelvis:

* Lagra data med Redis.
* Använda helmet.

# Redis

**3a** Sätt in ett värde med nyckeln favoriteColor via kommandotolken (CLI). Hämta sedan detta värdet

**3b** Gör samma sak som i 3a, fast via Node.

# Hash and Salt

**4** Gör en hemsida som har ett formulär med användarnamn och lösenord. Formuläret skickar en POST-request till /login. Lägg också till en sida i adressen /protected.

1. Lägg till ett hårdkodat användarnamn och lösenord i /login i express. Om användaren skickar med rätt inlogg ska den nu ha tillgång till /protected. Se också till att användare som inte är inloggade inte har tillgång till /protected
2. Lagra lösenorden i en databas istället för i koden.
3. Lägg till möjligheten för en användare att registrera sig på sidan.
4. Förhindra att användare registrerar sig med ett upptaget användarnamn. Kolla gärna in (<https://redis.io/commands/exists/>).
5. Istället för att lagra lösenorden i klartext, lagra lösenorden hashade och saltade med bcrypt.
6. Sätt krav på användaren att den måste ha minst 8 tecken i lösenordet.
7. Sätt ytterligare krav att lösenordet måste innehålla minst ett specialtecken och en siffra. *Tips: skriv egna funktioner som avgör detta*
8. Förhindra att formuläret redirectar till /login utan bara skickar en reqeust till /login
9. Lägg till att välkomsttexten istället är “Välkommen {användarnamn}!”
10. *Svår: Lägg till ett extra fält i databasen som avgör om en användare är admin eller inte (*[*https://redis.io/docs/data-types/hashes/*](https://redis.io/docs/data-types/hashes/)*). En admin ser texten välkommen admin!*

# JWT

**5.1** Gör en JWT med header { "alg": "HS256", "typ": "JWT"}, payload {"school": "jensen", "course": "web security"} och nyckeln: “secret” på <https://jwt.io/>

**5.2** Gör nu samma JWT utan JWT.io och endast med <https://www.devglan.com/online-tools/hmac-sha256-online> och <https://www.base64decode.org/>

**5.3** Implementera ett tokenbaserat inloggningssystem. Det ska finnas en inloggningssida, en sida tillgänglig endast för vanliga inloggade användare och en för admins.

# Miljövariabler och RBAC

**6.1** Byt ut dina hårdkodade lösenord till miljövariabler i ett tidigare projekt.

**6.2** Skapa en uppsättning av valfria endpoints. Du kan använda ett tidigare projekt om du vill. Se till att vissa av dessa endpoints bara är tillgängliga för användare med behörigheten “superuser”. Tänk på att koppla auktoriseringen till behörigheter och inte roller.

**6.3** Begränsa dina endpoints ytterligare så att man endast kan komma åt dem mellan vissa klockslag.

**6.4** Stäng av IP-adresser som har gjort mer än fem felaktiga requests från samtliga endpoints. Vill du göra det mer avancerat kan du ändra gränsen till max fem requests i ett 30-sekunders intervall. Kolla gärna in <https://www.abstractapi.com/guides/node-js-get-ip-address>

# TLS

**7.1** I följande övning ska vi ta en hemsida live med HTTPS. Övningen kommer också kräva att du betalar för en domän och en server. En domän kan exempelvis köpas via: [www.loopia.se](http://www.loopia.se) och en server via [www.digitalocean.com](http://www.digitalocean.com).

1. Välj ut en applikation som du vill ta till produktion.
2. Skapa en linuxserver. Jag körde på Ubuntu 22.04.<https://docs.digitalocean.com/products/droplets/how-to/create/> Välj också password som autentiseringsmetod.
3. Se till att din domän pekar på din nya servers IP. Exempelvis: <https://support.loopia.se/wiki/dnseditorn-a-och-cname/>. Detta tar ofta några timmar att slå igenom. Alla världens DNS-servrar måste nu uppdatera sig för att din domän ska peka på din servers IP.
4. Anslut till din server via kommandot: ssh root@<din domän>.
5. Du är nu inne på din server. Vi behöver först sätta upp en produktionsserver. Express devserver räcker alltså inte. <https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-22-04>
6. Vi ska nu fixa ett TLS-certifikat. <https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-22-04>
7. Installera Node. <https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-22-04>
8. Säg åt Nginx att omdirigera requests till Express. Du behöver inte göra första steget då du redan har en applikation. <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-22-04>
9. Success!

# OAuth 2.0

**8.1** Testa lite olika flows på: <https://www.oauth.com/playground/>

**8.2** Implementera ett OAuth flow med hjälp av GitHubs OAuth app. Du kan använda en tidigare applikation.

**8.3** Gör övningarna här: <https://oauth.school/>

**8.4** Sätt upp en gratis applikation på <https://auth0.com/>.

# WebSockets

**9.1** Gör en code-along med https://www.youtube.com/watch?v=rxzOqP9YwmM&ab\_channel=WebDevSimplified

**9.2** Uppgiften är att lösa en tidigare inlämningsuppgift för denna kursen. Jag har rensat bort en del krav för att minska scopet.

Uppgift

För att demonstrera att du uppfyller kursens mål, ska du bygga en säker fullstack

webbapp, med hjälp av teknikerna vi har gått igenom i kursen. Det ska vara en app för

säker chat.

Krav / features

1. [G] Backend ska ha en databas som sparar användare och meddelanden.

3. [G] Inloggning med användarnamn och lösenord, samt registrera ny användare.

5. [G] När man har loggat in, ska appen komma ihåg användaren med hjälp av JWT.

Detta innebär att om man laddar om sidan ska appen starta i inloggat läge.

6. [G] Appen ska innehålla kanalerna "public" och "private" (kolla in: <https://socket.io/docs/v4/tutorial/api-overview#rooms>)

9. [G] När användaren valt en kanal kan man skicka meddelanden i den.

10. [G] Det ska framgå på varje meddelande vilken användare som skickat det, samt

vid vilken tid. (dagar, timme, minut, sekund)

# XSS och CSRF

<https://portswigger.net/web-security/all-labs#cross-site-request-forgery-csrf>

https://portswigger.net/web-security/all-labs#cross-site-scripting