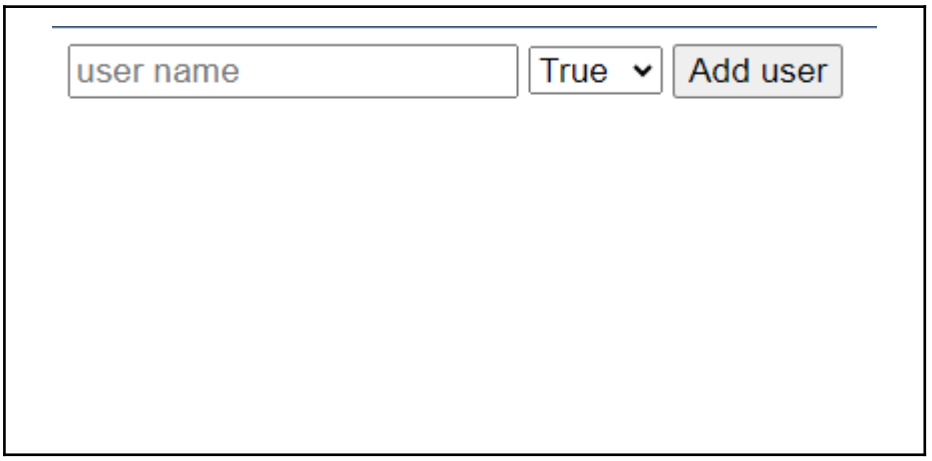
En enkel webbapplikation med både en frontend och en backend.

`Detta är en backend som handlar om **user requests** (get users, post user) och en frontend för att hämta alla users och för att kunna lägga till user.`

Syftet:

- ❖ Filhantering: Hur man hanterar data och filer i Node.js. Den visar hur du kan använda Node.js för att läsa och skriva **JSON-data till och från en fil**.
- ❖ API-hantering: förstå hur enkla **API-förfrågningar och svar** fungerar
- ❖ Om du behöver bygga en webbapplikation där användare kan **registrera sig, logga in och interagera med data på servern**, kan du använda denna koden som en startpunkt.
- ❖ Det är också användbart för att lära sig grunderna i att **hantera HTTP-requests, databasintegration och serverhantering med Node.js och Express.js.**



- **Backend (Node.js-server):**

- Vi skapade en Node.js-server med hjälp av Express.js, som är ett populärt webbramverk för att bygga webbapplikationer och API:er.
- Servern lyssnar på specifika HTTP-requests och svarar på dem. I vårt fall hanterar den GET- och POST-förfrågningar.

- **Databashantering:**

- Vi använde en JSON-fil som en enkel "databas" för att lagra användardata. Denna data är tänkt att vara ett exempel på information som du skulle kunna spara i en riktig databas i en mer komplex applikation.
- Vi använde Node.js "fs" -modulen för att läsa data från och skriva data till JSON-filen. Det här är en övning för att visa hur man kan hantera filer och data i Node.js.

- **GET Users Endpoint:**

- Vi skapade en GET-endpoint på vår server som tillåter frontend att hämta användardata. Frontenden kan skicka en GET-förfrågan till servern för att begära en lista över användare eller filtrera användare baserat på ett namn.

- **POST User Endpoint:**

- Vi skapade också en POST-endpoint som tillåter frontend att skicka nya användaruppgifter till servern för att lägga till en ny användare i vår "databas".

- **Frontend (HTML och JavaScript):**

- Frontend-delen består av en enkel HTML-sida och JavaScript-kod.
- HTML-sidan har ett formulär där användaren kan skriva in namn och välja om användaren är en administratör eller inte. När formuläret skickas, använder JavaScript Fetch API för att skicka en POST-förfrågan till vår backend.

I den här uppgiften används ett enkelt API som implementeras med hjälp av Express.js för att hantera GET- och POST-förfrågningar.

GET-förfrågningar:

- Endpoint: `/users` är det endpoint som används för **GET-förfrågningar**.

Det innebär att när en klient gör en GET-förfrågan till `http://localhost:3000/users`, så kommer den att nå den här endpointen.

- Syfte: **GET-förfrågningar används för att hämta data från servern**. I det här fallet hämtar vi användarinformation från JSON-filen **users.json**.

Om man går till http://localhost:3000/users, eller om du använder JavaScript i frontend-koden för att göra en GET-förfrågan till samma URL, kommer du att få tillbaka en lista med användare som svar.

POST-förfrågningar:

- Endpoint: /users är också används för POST-förfrågningar. Det innebär att när en klient skickar data till http://localhost:3000/users via en **POST-förfrågan**, kommer den att nå den här endpointen.
- Syfte: **POST-förfrågningar används för att skicka data till servern för att skapa något nytt. I detta fall skickar vi användarinformation (namn och admin-status) till servern för att skapa en ny användare.**

Om du har ett formulär på din frontend som låter användare fylla i sitt namn och välja om de är admin eller inte, kan du använda JavaScript för att skapa en POST-förfrågan till http://localhost:3000/users med den här informationen. Servern kommer då att behandla förfrågan och skapa en ny användare.

Sammanfattningsvis används API-endpunkterna **/users** för att hantera **`läsning (GET)`** och **`skrivning (POST)`** av **användarinformation**. GET används för att hämta information från servern, medan POST används för att skicka information till servern för att skapa nya poster. Dessa förfrågningar används för att **interagera med och manipulera data på servern**.

API-endpunkter, ibland bara kallade "endpunkter", är specifika URL-adresser eller resursvägar som en webbserver tillhandahåller för att låta klientapplikationer kommunicera med servern. De fungerar som gränssnitt för att utföra olika operationer eller hämta data från servern.

En Express.js-applikation är en webbapplikation som utvecklats med hjälp av Express.js, som är ett minimalt och flexibelt Node.js-webbapplikationsramverk. Express.js gör det enkelt att bygga webbapplikationer och API:er genom att erbjuda ett lager med användbara funktioner och hjälpa utvecklare att hantera HTTP-requests, routing, middleware och mycket mer.

Här är några nyckelaspekter och komponenter i en Express.js-applikation:

1. Middleware: Express.js tillåter användning av middleware, som är funktioner som kör före eller efter en förfrågan hanteras av servern. Middleware kan användas för saker som autentisering, CORS (Cross-Origin Resource Sharing)-hantering, loggning och dataformatering.

2. Routing: Express.js gör det enkelt att definiera och hantera routes. Routes definierar URL-sökvägar och de funktioner som ska köras när en specifik sökväg anropas. Till exempel kan du definiera en route för att hämta användarinformation när en användare går till "/users".

3. HTTP-requests och -svar: Du kan hantera olika typer av HTTP-requests som GET, POST, PUT och DELETE med Express.js. Dessutom kan du svara på förfrågningar med JSON-data, HTML-sidor eller andra typer av svar.

4. Databashantering: Även om Express.js inte inkluderar något specifikt databashanteringslager, kan du använda det med databasramverk som Mongoose (för MongoDB), Sequelize (för SQL-databaser) eller andra databasmoduler för att ansluta till och hantera din databas.

5. Middlewares från tredje part: Det finns ett stort ekosystem av Express.js-middlewares från tredje part som kan användas för att lägga till extra funktionalitet i din applikation. Till exempel kan du använda middlewares för autentisering, sessionshantering och säkerhet.

Sammanfattningsvis är en Express.js-applikation en Node.js-applikation som använder Express.js-ramverket för att snabbt och effektivt bygga webbapplikationer och API:er. Det

ger en strukturerad och organiserad väg att skapa webbapplikationer och hantera HTTP-requests och -svar.