I skal lave en kontaktbog.

Der er 20 opgaver, fordelt på 4 grupper, og hver opgave udløser op til 5 point. Hvis man ikke kan løse en opgave kan man i stedet skrive hvad man ville gøre for at opnå en delmængde af de tilgængelige point. Hver af de 20 opgaver giver 0-5 point, så der i alt kan opnås 0-100 point.

Alle opgaver skal besvares i dette dokument som efterfølgende afleveres i word- eller pdf format.

Opgaverne er uafhængige af hinanden, så hvis en opgave besvares forkert har det ikke indflydelse på andre besvarelser.

Hvis en opgave besvares helt korrekt, giver den 5 point, og hvis der ikke er noget svar giver den 0 point. Hvis der er en forklaring eller et mangelfuldt svar, vurderes antal givne point ud fra den enkelte besvarelse omfang og rigtighed.

Hold besvarelserne simple så de KUN besvarer spørgsmålet. Hvis koden gør andet og/eller mere end det efterspurgte, så vil dette tolkes som misforståelse af svar og/eller spørgsmål og trække point fra det givne spørgsmål. Eksempelvis vil et spørgsmål som kan besvares med 2 linjer kode betragtes som værende besvaret forkert hvis der besvares med 50 linjer kode, selv hvis de 2 korrekte linjer er indeholdt i besvarelsen.

Den endelige besvarelse er kun de svar som er anført i dette dokument. Der kan ikke vedhæftes kodefiler.

Gennemgå hele opgavesættet før påbegyndelse, og start med at løse de opgaver som er nemmest for dig.

Der er fuld internetadgang under hele prøven, men der må ikke anvendes generativ AI eller kommunikation.

1. Data (MongoDB & JSON)

Du har en collection, contacts, som har denne generelle struktur:

```
{
  "_id": ObjectId("..."),
  "first_name": "John",
  "last_name": "Doe",
  "phones": [
        { "type": "work", "number": "+45 98 45 67 89" },
        { "type": "mobile", "number": "+45 87 65 43 21" }
  ],
  "email": “jd@sdu.dk”,
  "created_at": ISODate("2025-01-01T10:00:00Z")
}
```

a) Updater en contact i kollektionen

En af dine kontakter har fået nyt arbejdstelefonnummer.
Skriv en MongoDB-forespørgsel, der opdaterer telefonnummeret af typen ”work” til det følgnde i kollektionen contacts:

- "+45 76 41 37 09"

SVAR

```
db.users.updateOne(

  { _id: ObjectId("..."), "phones.type": "work" },

  { $set: { "phones.$.number": "+45 76 41 37 09” } }

)
```

b) Slet contacts som er mere end 5 år gamle

Brugeren vil rydde op i sine kontakter og ønsker at slette alle kontakter som er mere end 5 år gamle.
Skriv en MongoDB-forespørgsel, der sletter alle kontakter med ”created_at” ældre end 5 år.

SVAR

```
const fiveYearsAgo = new Date();
```

```
fiveYearsAgo.setFullYear(fiveYearsAgo.getFullYear() - 5);


db.contacts.deleteMany({
  created_at: { $lt: fiveYearsAgo }
});
```

c) Hent en liste over kontakter der starter med J

Brugeren foretager en søgning i contacts og vil gerne se alle kontakter hvis fornavn starter med J.
Skriv en MongoDB-forespørgsel, der henter alle kontakter der starter med J.

SVAR

```
db.contacts.aggregate([
{$match:{first_name:{$regex: /^J/}}}
])
```

d) Slet kollektionen contacts

En bruger har besluttet at afinstallere applikation og derved skal kollektionen contacts slettes.
Skriv en MongoDB-forespørgsel, der sletter kollektionen contacts.

```
db.contacts.drop()
```

e) Tæl hvor mange kontakter der er i kollektionen

Brugerne vil gerne se hvor mange kontakter der i applikationen.
Skriv en MongoDB-forespørgsel, der tæller antallet af kontakter i kollektionen og viser resultatet som følgende:

```
[ { "Total contacts": XX } ]
```

SVAR

```
db.contacts.aggregate([
 {
  $count: "Total contacts"
 }
```

])

2. NodeJS

Du skal oprette endpoints i Node.js med Express, som en React-applikation senere vil bruge.

a) Opret endpoint til at hente alle contacts sorteret alfabetisk

Skriv et Express-endpoint, der håndterer GET-forespørgsler til:
GET /api/contacts

- Endpointet skal hente alle kontakter fra kollektionen contacts sorteret i alfabetisk rækkefølge efter fornavn og i tilfælde af at to kontakter har samme fornavn sorteres der derefter på efternavn.
- Returner noterne som JSON.

SVAR

```
app.get("/api/contacts", async (req, res) => {

  const contacts = await Contact.find().sort({

    firstName: 1,

    lastName: 1

  });



  res.json(contacts);

});
```

b) Opret endpoint til at tilføje en ny kontakt

Skriv et Express-endpoint, der håndterer POST-forespørgsler til:
POST /api/contacts

Endpointet skal modtage følgende data i request-body (som JSON):

```
{
  "first_name": "Lars",
  "last_name": "Larsen",
  "email": "lars@jysk.dk",
  "phones": [{”type”: ”work”, ”number”: ”+45 12 34 56 78”}]
}
```

Endpointet skal:

- Tilføje created_at som den nuværende dato.
- Indsætte kontakten i kollektionen contacts.
- Returnere den oprettede note som JSON.

SVAR

```
app.post("/api/contacts", async (req, res) => {

  const contact = {

    first_name: req.body.first_name,

    last_name: req.body.last_name,

    email: req.body.email,

    phones: req.body.phones,

    created_at: new Date()

  };

  const newContact = await Contact.create(contact);

  res.json(newContact);

});
```

c) Opret endpoint til at opdatere en kontakt

Skriv et Express-endpoint, der håndterer PUT-forespørgsler til:
PUT /api/contacts/:id

- Endpointet skal opdatere en specifik kontakt baseret på dens _id.
- Det modtager følgende data i request-body (som JSON):

```
{
  "last_name": "Jacobsen",
  "email": "jacobsen@mail.dk",
  “phones”: [
            {”type”: ”work”, ”number”: ”+45 62 84 56 78”},
            {”type”: ”mobile”, ”number”: ”+45 54 34 56 91”}
    ]
}
```

Endpointet skal:

- Opdatere last_name, email og phones.
- Returnere den opdaterede note som JSON.

SVAR

```
app.put("/api/contacts/:id", async (req, res) => {

  const updatedContact = await Contact.findByIdAndUpdate(

    req.params.id,

    {

      last_name: req.body.last_name,

      email: req.body.email,

      phones: req.body.phones

    },

    { new: true }

  );
```

```
  res.json(updatedContact);

});
```

d) Opret endpoint til at slette en specifik kontakt

Skriv et Express-endpoint, der håndterer DELETE-forespørgsler til:
DELETE /api/contacts/:id

- Endpointet skal:
    1. Slette kontakten med det _id, der er angivet som URL-parameter.
    2. Returnere en besked som JSON:

{"message": "The contact “first_name last_name” was deleted successfully"}

SVAR

```
app.delete("/api/contacts/:id", async (req, res) => {

  const contact = await Contact.findByIdAndDelete(req.params.id);



  res.json({

    message: `The contact "${contact.first_name} ${contact.last_name}" was deleted successfully`

  });

});
```

e) Opret endpoint til at søge efter noter baseret på tags

Skriv et Express-endpoint, der håndterer GET-forespørgsler til:
GET /api/contacts/search?type=work

Endpointet skal hente alle kontakter, der indeholder den angivne type af telefon i feltet phones og returnere kontakterne som JSON.

SVAR

```
app.get("/api/contacts/search", async (req, res) => {

  const contacts = await Contact.find({

    "phones.type": req.query.type

  });



  res.json(contacts);

});
```

3. REACT

a) Vis kontakter i en liste

Lav en komponent, der henter og viser alle kontakter alfabetisk i en liste

- Brug useEffect til at hente data fra endpointet GET /api/contacts
- Brug useState til at gemme listen af contacts.
- Vis kontakterne i liste, inddelt i sektioner efter forbogstav.

SVAR

```
import { useEffect, useState } from "react";

function ContactList() {
  const [contacts, setContacts] = useState([]);


  useEffect(() => {
    fetch("/api/contacts")
      .then(res => res.json())
      .then(data => setContacts(data));
  }, []);
 return (
    <div>
      {contacts.map(contact => (
        <div key={contact._id}>
          {contact.first_name} {contact.last_name}
        </div>
      ))}
    </div>
  );
}
export default ContactList;
```

b) Opret en form til at tilføje en ny kontakt

Lav en komponent, hvor brugeren kan tilføje en ny kontakt via en form.

Formen skal have felter til fornavn, efternavn, email, arbejdstelefon og privattelefon

Ved indsendelse af formen skal data sendes til endpointet POST /api/contacts.

SVAR

```
function AddContact() {
  const handleSubmit = (e) => {
    e.preventDefault();

    const form = e.target;

    fetch("/api/contacts", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        first_name: form.first_name.value,
        last_name: form.last_name.value,
        email: form.email.value,
        phones: [
          { type: "work", number: form.work_phone.value },
          { type: "private", number: form.private_phone.value }
        ]
```

```
    })
  });
};

return (
  <form onSubmit={handleSubmit}>
    <input name="first_name" placeholder="Fornavn" />
    <input name="last_name" placeholder="Efternavn" />
    <input name="email" placeholder="Email" />
    <input name="work_phone" placeholder="Arbejdstelefon" />
    <input name="private_phone" placeholder="Privattelefon" />
    <button type="submit">Gem</button>
  </form>
);
}

export default AddContact;
```

c) Lav en side der viser en kontakt detaljeret.

Lav en komponent, der viser en detaljeret kontakt.

Komponentet skal hente data fra endpointet GET /api/contacts/:id, hvor id er kontakt-id'et.

Vis alle kontakt-detaljer (Fornavn, efternavn, email og alle telefonnumre)

SVAR

```
import { useEffect, useState } from "react";


function ContactDetails({ id }) {
  const [contact, setContact] = useState(null);


  useEffect(() => {
    fetch(`/api/contacts/${id}`)
      .then((res) => res.json())
      .then((data) => setContact(data));
  }, [id]);


  if (!contact) return null;


  return (
    <div>
      <div>{contact.first_name} {contact.last_name}</div>
      <div>{contact.email}</div>


      {contact.phones.map((p, i) => (
        <div key={i}>
          {p.type}: {p.number}
        </div>
```

```
      ))}
    </div>
  );
}

export default ContactDetails;
```

d) Lav en funktion til at slette en kontakt

Tilføj en slet-knap i den detaljerede visning af en kontakt.

Knappen skal sende trigger en alert hvor man kan bekræfte sletningen og ved bekræftelse sende en DELETE-forespørgsel til endpointet DELETE /api/contacts/:id.

SVAR

```
function ContactDetails({ id }) {


  const deleteContact = () => {
    if (window.confirm("Er du sikker på, at du vil slette kontakten?")) {
      fetch(`/api/contacts/${id}`, {
        method: "DELETE"
      });
    }
  };
```

```
  return (
    <div>
      <button onClick={deleteContact}>Slet kontakt</button>
    </div>
  );
}
```

e) Tilføj en filtreringsfunktion til listevisningen

Tilføj en filtreringsfunktion, så brugeren kan filtrere kontakter baseret på telefontype

Tilføj en dropdown over listevisningen hvor man kan vælge mellem typerne ”work” og ”mobile”.

Når brugeren trykker på "Søg", skal data hentes fra endpointet

SVAR

GET /api/contacts/search?type=<type>.

```
import { useState } from "react";

function ContactFilter({ setContacts }) {
  const [type, setType] = useState("work");

  const search = () => {
```

```
    fetch(`/api/contacts/search?type=${type}`)
      .then((res) => res.json())
      .then((data) => setContacts(data));
  };

  return (
    <div>
      <select value={type} onChange={(e) => setType(e.target.value)}>
        <option value="work">work</option>
        <option value="mobile">mobile</option>
      </select>

      <button onClick={search}>Søg</button>
    </div>
  );
}

export default ContactFilter;
```

4. Cyber Security

I disse opgaver skal du bare aflevere et enkelt eksempel på det forespurgte og ikke en fuld opsætning på al din kode. Men det skal være et eksempel fra noget der er lavet i forbindelse med indeværende eksamen. Du kan vælge at bruge de nævnte pakker i de enkelte opgaver eller selv at lave en simpel version som viser din forståelse.

a) Brute force angreb

SVAR

Hvad er et brute force angreb og hvordan kan man undgå det?

Et brute force-angreb er et angreb, hvor en hacker systematisk afprøver mange forskellige passwords for at få adgang. Det kan forebygges ved at begrænse login-forsøg, bruge stærke passwords og anvende rate limiting eller to-faktor-godkendelse.

b) Brug en API key

Kald en vilkårlig API (eksempel.dk), men anvend en API-key (minapikey123456). Vis hvordan du laver kaldet her.

SVAR

```
fetch("https://eksempel.dk/api/data", {
  headers: {
    "Authorization": "Bearer minapikey123456"
  }
})
  .then(res => res.json())
  .then(data => console.log(data));
```

c) Password Hashing

Dine passwords er altid hashede i databasen. Skriv en funktion som tjekker et password som er sendt ind i et post request fra et input med name=”password” . Kontroller om det postede password er korrekt i forhold til det hashede password som er gemt i databasen.

SVAR

```
const bcrypt = require("bcrypt");

async function checkPassword(req, hashedPasswordFromDB) {
  const inputPassword = req.body.password;

  const isMatch = await bcrypt.compare(inputPassword, hashedPasswordFromDB);
```

```
  return isMatch;
}
```

d) Hashing og kryptering

Hvad er forskellen på hashing og kryptering, og hvorfor hasher vi passwords i stedet for at kryptere dem?

SVAR

Hashing er en envejs-proces, hvor data ikke kan genskabes til original form.
Kryptering er tovejs og kan dekrypteres igen med en nøgle.
Passwords hashes, fordi de ikke skal kunne genskabes, selv hvis databasen bliver kompromitteret.
Hashing er derfor mere sikkert end kryptering til passwords.

e) Audit logging

For at overvåge databasen for mistænkelig aktivitet skal du logge visse sikkerhedsrelaterede hændelser. Lav en auditlog som holder øje med ændringer i databasen, uafhængigt af den tilknyttede applikation.

SVAR

```
const { MongoClient } = require("mongodb");

async function startAuditLog() {
  const client = new MongoClient("mongodb://localhost:27017");
  await client.connect();

  const db = client.db("mydb");
  const contacts = db.collection("contacts");
  const auditlog = db.collection("auditlog");

  contacts.watch().on("change", (change) => {
    auditlog.insertOne({
      collection: "contacts",
      operation: change.operationType,     // insert/update/delete
      documentId: change.documentKey?._id,
      changedAt: new Date(),
      change
```

```
    });
  });
}

startAuditLog();
```