1. Data (MongoDB & JSON)

Update element inside collection

```
db.users.updateOne(
  { _id: ObjectId("64d1b840e948f5e1d7c8a912"), "phones.type": "work"},
  { $set: { "phones.$.number": "+45 76 41 37 09” } }
)
```

Update element to new value

```
db.notes.updateMany(
{ title: {$in:[“”, null]} },
{ $set: { title: “Untitled Note” }
})
```

Add element to collection

```
db.notes.insertOne(
{
user_id: ObjectId(“64d1b840e948f5e1d7c8a912”),
title: “Indkøbsliste“,
content: "Mælk, brød, æg, smør",
tags: ["indkøb", "personlig"],
created_at: new Date()
updated_at: new Date()
})
```

Delete element using filter

```
const fiveYearsAgo = new Date();
db.contacts.deleteMany({
  created_at: { $lt: fiveYearsAgo.getFullYear() - 5 }
});
```

Delete elements with condition

```
db.notes.deleteMany(
{tags: {$eq: ”arbejde”}}
)
```

Delete all values by ID

```
db.notes.deleteMany({
user_id: {$eq: ObjectId(“64d1b840e948f5e1d7c8a912”)
})
```

Get element

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

Delete collection

db.contacts.drop()

Count elements in a collection

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

Count amount of repeating elements

```
db.notes.aggregate([
  { $unwind: "$tags" },
  { $group: { _id: "$tags", count: { $sum: 1 } } },
  { $project: { _id: 0, tag: "$_id", count: 1 } }
])
```

2. NodeJS

Create get endpoint and sort

```
app.get("/api/contacts", async (req, res) => {
  const contacts = await Contact.find().sort({
    firstName: 1,
    lastName: 1
  });

  res.json(contacts);
});
```

Create endpoint to get all elements for specific unit

```
app.get("/api/notes/:user_id", async (req, res) => {
const notes = await Note.find({user_id: req.params.user_id});
res.json(notes);
};
```

Create endpoint to add new element

```
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);
});
```

```
app.post("/api/notes", async (req, res) => {
  const note = {
    user_id: req.body.user_id,
    title: req.body.title,
    content: req.body.content,
    tags: req.body.tags,
    created_at: new Date(),
    updated_at: new Date()
  };
  const newNote = await Note.create(newNote );
  res.json(newNote );
});
```

Create endpoint to update element

```
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);
});
```

```
app.put("/api/notes/:id", async (req, res) => {
  const updatedNote = await Note.findByIdAndUpdate(
    req.params.id,
    {
  title: req.body.title,
  tags: req.body.tags,
  content: req.body.content,
updated_at: new Date()
    },
    { new: true }
  );
  res.json(updatedNote);
});
```

Create endpoint to delete specific element

```
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`
  });
});
```

```
app.delete("/api/notes/:id", async (req, res) => {
  const contact = await Contact.findByIdAndDelete(req.params.id);
  res.json({
    message: “Note deleted successfully”
  });
});
```

Create endpoint to search for elements by condition

```
app.get("/api/contacts/search", async (req, res) => {
  const contacts = await Contact.find({
    "phones.type": req.query.type
  });

  res.json(contacts);
});
```

3. REACT

Show a list of elements

```
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;
```

```
import { useEffect, useState } from "react";
function NoteList({userId}) {
  const [notes, setNotes] = useState([]);

  useEffect(() => {
    fetch("/api/notes/${userId}")
      .then(res => res.json())
      .then(data => setNotes(data));
  }, [userId]);
 return (
    <div>
      {notes.map(note => (
        <div key={note._id}>
          Titel: {note.title} ; Dato: {note.created_at}
        </div>
      ))}
    </div>
  );
}
export default NotesList;
```

Create form to add new element

```
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;
```

```
function AddNote() {
  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");
  const [tags, setTags] = useState("");
const submit = e => {
    e.preventDefault();
    fetch("/api/notes", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        user_id: "12345",
        title,
        content,
        tags: tags.split(",")
      })
    });
  };
  return (
    <form onSubmit={submit}>
      <input onChange={e => setTitle(e.target.value)} />
      <textarea onChange={e => setContent(e.target.value)} />
      <input onChange={e => setTags(e.target.value)} />
      <button>Gem</button>
    </form>
  );
}
```

Create side that shows elements

```
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;
```

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

function NoteDetail({ id }) {
  const [note, setNote] = useState(null);
  useEffect(() => {
    fetch(`/api/notes/${id}`)
      .then(res => res.json())
      .then(data => setNote(data));
  }, [id]);

  if (!note) return null;

  return (
    <div>
      <h2>{note.title}</h2>
      <p>{note.content}</p>
      <p>Tags: {note.tags.join(", ")}</p>
      <p>Dato: {note.created_at}</p>
    </div>
  );
}
```

Create function to delete element

```
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>
  );
}
```

```
function NoteDetail({ note }) {
  const deleteNote = () => {
    fetch(`/api/notes/${note._id}`, { method: "DELETE" });
  };

  return <button onClick={deleteNote}>Slet</button>;
}
```

Add filtering functionality for list view

```
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;
```

Add search functionality to list view

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

function NoteSearch({ setNotes }) {
  const [tag, setTag] = useState("");

  const search = async () => {
    const res = await fetch(`/api/notes/search?tag=${tag}`);
    const data = await res.json();
    setNotes(data);
  };

  return (
    <>
      <input onChange={e => setTag(e.target.value)} />
      <button onClick={search}>Søg</button>
    </>
  );
}
```

4. Cyber Security

What is brute force attack?

A brute-force attack is an attack in which a hacker systematically tries many different passwords to gain access. It can be prevented by limiting login attempts, using strong passwords, and applying rate limiting or two-factor authentication.

Use an API key

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

Password Hashing

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

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

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

  return isMatch;
}
```

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

async function hashPassword(password) {
  return await bcrypt.hash(password, 10);
}
```

What is the difference between hashing and encryption, and why do we hash passwords instead of encrypting them?

Hashing is a one-way process where data cannot be restored to its original form.
Encryption is a two-way process and can be decrypted again using a key.
Passwords are hashed because they should not be recoverable, even if the database is compromised. Hashing is therefore more secure than encryption for passwords.

Audit logging

```
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();
```

Prevent MongoDB injection attack

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

function validateUserId(user_id) {
  if (!ObjectId.isValid(user_id)) {
    throw new Error("Invalid user_id");
  }
  return user_id;
}
```

Use Basic auth to create GET request

```
function getSecureData() {
  fetch("/api/secure", {
    headers: {
      Authorization: "Basic " + btoa("sdu:exam")
    }
  });
}
```

Protect against Cross-Site Request Forgery (XSRF)

The React client retrieves a CSRF token from the server (e.g. via a GET endpoint). The token is stored temporarily on the client (e.g. in state). For POST/PUT/DELETE requests, React sends the token with the request (typically in a header such as X-CSRF-Token). The server validates the token and rejects the request if the token is missing or invalid.

```
const express = require("express");
const cookieParser = require("cookie-parser");
const csurf = require("csurf");

const app = express();

app.use(express.json());
app.use(cookieParser());

// CSRF middleware (double-submit cookie style)
const csrfProtection = csurf({ cookie: true });

// Endpoint til at hente et token
app.get("/api/csrf-token", csrfProtection, (req, res) => {
  res.json({ csrfToken: req.csrfToken() });
});

// Beskyt state-changing routes
app.post("/api/notes", csrfProtection, (req, res) => {
  res.json({ ok: true });
});
```

Logging off security handling

```
const winston = require("winston");
const { ObjectId } = require("mongodb");

const logger = winston.createLogger({
  level: "warn",
  transports: [new winston.transports.Console()]
});

function validateUserId(user_id) {
  if (!ObjectId.isValid(user_id)) {
    logger.warn("Unauthorized access attempt: invalid user_id", { user_id });
    throw new Error("Invalid user_id");
  }
}
```