2021 - A

```
/* Jeg valgte å skrive koden i TypeScript slik at jeg kan dra nytte av den
statiske typesjekkingen.
Dette gjør koden mer oversiktlig, og det blir lettere for "debugging" siden man
ser om variabler har type-feil.
Jeg har kalt variablene etter det ønskede navnet på norsk siden dette ble etterspurt.
Siden jeg personlig foretrekker å programmere alt på engelsk er derfor bl.a.
metodene skrevet på engelsk også.
*/

// Defining the class for a vehicle
class Kjoretoy {
    // Declaring private variables according to task
    #maksHastighet: number;
    #kjorelengde: number;
    #farge: string = "hvit";

    constructor(speedLimit: number, distance: number) {
        this.#maksHastighet = speedLimit;
        this.#kjorelengde = distance;
    }

    get speedLimit() {
        return this.#maksHastighet;
    }

    get distance() {
        return this.#kjorelengde;
    }
}

// Creating an instance "car" for the vehicle class
const car: Kjoretoy = new Kjoretoy(160, 100);

// Logging information about the vehicle to the console
console.log(`Kjøretøyet har en maksimal hastighet på ${car.speedLimit}
km/t og kjører en distanse på ${car.distance}km`);


// Defining the "sub-class" Buss which inherits properties from Kjoretoy
class Buss extends Kjoretoy {
    // Declaring private variable for class
    #maksPassasjerer: number;

    constructor(maksHastighet: number, kjorelengde: number, passangerLimit: number) {
        super(maksHastighet, kjorelengde);
```

```
        this.#maksPassasjerer = passangerLimit;
    }

    sjekkAntall(antallpassasjerer) {
        return (antallpassasjerer > this.#maksPassasjerer) ? true : false;
    }

    getRentalPrice() {
        return (this.#maksPassasjerer * 100) * 1.25;
    }
}

// Creating an instance "bus" for the Buss class
const bus: Buss = new Buss(120, 300, 60);

// Using the defined method to check if 50 passangers is too many for this particular bus
console.log(`Vil 50 passasjerer være for mange for bussens kapasitet?
${bus.sjekkAntall(50)}`);

// Using the method getRentalPrice() to find the correct price for renting
//this particular bus
console.log(`Pris for leie av bussen vil komme på ${bus.getRentalPrice()}kr`);
```

```
/*
Slik som i oppgave 1 har jeg valgt å benytte meg av TypeScript for å lettere
kunne kontrollere at variabler er av riktig sort.
Jeg har også brukt Promises for å kontakte databasen: Merk at i ".catch"
methodene har jeg ikke inkludert (error) argumentet,
selv om det blir sendt videre med "reject(error)" i ChatService.
Dette er fordi jeg har tenkt mer på bruker-feedback,
enn selve utviklingsprosessen. Ettersom dette skal være et "ferdig" produkt
ser jeg ikke at det er nødvendig.
Hvis ikke kan man bare erstatte ".catch(*)" med ".catch(error => c
onsole.error(error))" f.eks. om en ønsker å debugge.

Jeg antar også at variabler for database oppkoblinger er importert i
objektet "pool" fra filen "mysql-pool"
*/

import * as React from 'react';
import { Component } from 'react-simplified';
import ReactDOM from 'react-dom';
import { NavLink, HashRouter, Route } from 'react-router-dom';
import { Alert, Card, Row, Column, NavBar, Button, Form } from './widgets';
import { createHashHistory } from 'history';
import { pool } from './mysql-pool';

const history = createHashHistory();

// Defining a service class to contain all database related code
class ChatService {
  // Method that retrieves all chat rooms in the database
  getChatRooms() {
```

```
    return new Promise((resolve, reject) => {
      pool.query("SELECT * FROM ChatRooms", (error, response) => {
        if (error) return reject(error);

        resolve(response);
      });
    });
  }

  // Method that retrieves a specific chat room based on the chat room ID
  getChatRoomByID(id: number) {
    return new Promise<ChatRoom>((resolve, reject) => {
      pool.query("SELECT * FROM ChatRooms WHERE id=?", [id], (error, response) => {
        if (error) reject(error);

        resolve(response[0]);
      })
    })
  }

  // Method that retrieves all messages related to a particular chat room
  getMessages(chatRoomID: number) {
    return new Promise((resolve, reject) => {
      pool.query("SELECT * FROM Messages WHERE chatRoomId=?", [chatRoomID],
(error, response) => {
        if (error) return reject(error);

        resolve(response);
      });
    });
  }

  // Method that adds a chat room to the database
  createChatRoom(chatRoom: ChatRoom) {
    return new Promise((resolve, reject) => {
      pool.query("INSERT INTO ChatRooms (title, description) VALUES (?,?)",
[chatRoom.title, chatRoom.description], (error, response) => {
        if (error) reject(error);

        resolve(response);
      });
    });
  }

  // Method that adds a message to the message table in the database
  sendMessage(message: Message) {
    console.log(message);
    return new Promise((resolve, reject) => {
      pool.query("INSERT INTO Messages (text, chatRoomId) VALUES (?,?)",
[message.text, Number(message.chatRoomID)], (error, response) => {
        if (error) reject(error);

        resolve(response);
      })
    })
  }
}
```

```
// Creating an instance of the ChatService class
const chatService: ChatService = new ChatService;

// Defining a class for a ChatRoom
class ChatRoom {
  id?: number | string;
  title: string;
  description: string;

  constructor(title: string = "", description: string = "") {
    this.title = title;
    this.description = description;
  }
}


// Defining a class for messages
class Message {
  text: string;
  chatRoomID: number | string;

  constructor(text: string, chatRoomID = "") {
    this.text = text;
    this.chatRoomID = chatRoomID;
  }
}

// Creating the home page for the application
class Home extends Component {
  // Declaring an array to hold all chat rooms
  chatRooms: any[] = []; // This is set at type "all" because the objects are
//not instances of a class when they are retrieved from the database

  // Declaring an instance of a new chat room
  newChatRoom: ChatRoom = new ChatRoom();

  render() {
    if (!this.chatRooms) return null;

    return (
      <div>
        <Card title="ChatApp">
          <Card title="Chat Rooms">
            <ul style={{ listStyleType: "none" }}>
              {this.chatRooms.map((room) => {
                return (
                  <li key={room.id}>
                    <NavLink to={"/chatroom/" + room.id}>
                      {room.title}
                    </NavLink>
                  </li>
                )
              })}
            </ul>
          </Card>
          <Card title="New Chat Room">
            <Form.Label>Title:</Form.Label>
            <Form.Input type="text" value={this.newChatRoom.title} onChange={(event)
```

```
=> this.newChatRoom.title = event.currentTarget.value} />

            <Form.Label>Description:</Form.Label>
            <Form.Input type="text" value={this.newChatRoom.description}
onChange={(event) => this.newChatRoom.description = event.currentTarget.value} />

            <Button.Success onClick={this.add}>Create new room</Button.Success>
          </Card>
        </Card>
      </div>
    );
  }

  mounted() {
    chatService
      .getChatRooms()
      .then((rooms: any) => this.chatRooms = rooms)
      .catch(() => Alert.danger("ERROR! Failed to fetch chatrooms"));
  }

  add() {
    chatService
      .createChatRoom(this.newChatRoom)
      .then(() => {
        Alert.success("Added new chat room");
        this.mounted();
      })
      .catch(() => Alert.danger("ERROR! Failed to add new chat room"));
  }
}

// Creating the chat room page. This will dynamically load the wanted chat room
class ChatRoomPage extends Component<{ match: { params: { id: string } } }> {
  // Declaring instance of ChatRoom
  chatroom: ChatRoom = new ChatRoom();

  // Declaring an instance of a message where the chatRoomId is the
//currect chat room you're in
  newMessage: Message = new Message("", this.props.match.params.id);

  // Declaring an array to hold the messages
  messages: any[] = [];

  render() {
    if (!this.chatroom) return null;
    this.messages.map((msg) => console.log(msg))

    return (
      <div>
        <Card title="ChatApp">
          <Card title={this.chatroom.title}>
            {this.chatroom.description}
            <Card title="Messages">
              <ul style={{ listStyleType: "none" }}>
                {this.messages.map((msg) => {
                  return (
                    <li key={msg.id}>
                      {`>${msg.text}`}
```

```
                    </li>
                  )
                })}
              </ul>
            </Card>
          </Card>
          <Card title="New Message">
            <Form.Input type="text" onChange={(event) => this.newMessage.text =
event.currentTarget.value} />

            <Button.Success onClick={this.postMessage}>Create Message</Button.Success>
          </Card>
          <Button.Light onClick={() => history.push("/")}>Go to main menu</Button.Light>
        </Card>
      </div>
    );
  }

  mounted() {
    chatService
      .getChatRoomByID(Number(this.props.match.params.id))
      .then((chatroom: any) => this.chatroom = chatroom)
      .catch(() => Alert.danger("ERROR! Failed to fetch chatroom"));

    chatService
      .getMessages(Number(this.props.match.params.id))
      .then((messages: any) => this.messages = messages)
      .catch(() => Alert.danger("ERROR! Failed to retrieve messages"))
  }

  postMessage() {
    chatService
      .sendMessage(this.newMessage)
      .then(() => {
        Alert.success("Posted new message")
        this.mounted();
      })
      .catch(() => Alert.danger("ERROR! Failed to post message"));
  }
}

// Creating a hashRouter in the "ReactDOM.render" method
const root = document.getElementById('root');
if (root)
  ReactDOM.render(
    <HashRouter>
      <div>
        <Route exact path="/" component={Home} />
        <Route path="/chatroom/:id" component={ChatRoomPage} />
      </div>
    </HashRouter>,
    root
  );
```