Alternative løsninger fra Git

2018 - 1

```
class Land {
    constructor(navn, areal, folketall, toppdomene) {
        this.navn = navn;
        this.areal = areal;
        this.folketall = folketall;
        this.toppdomene = toppdomene;
    }
    getDensity() {
        return `${(this.folketall / this.areal).toFixed(2)} innbyggere pr. km²`;
    }
}

let Norge = new Land("Norge", 328899, 5385000, ".no");
let Sverige = new Land("Sverige", 449964, 9455000, ".se");
let Russland = new Land("Russland", 17075400, 143989000, ".ru");
let Kina = new Land("Kina", 9596960, 1409517000, ".cn");

let land = [Norge, Sverige, Russland, Kina];

land.map(land => {
    let info = document.createElement('p');
    info.innerText += `${land.navn} (${land.toppdomene}) har ${land.folketall}
                          innbyggere og et areal på ${land.areal}km².
                       ${land.navn} har en densitet på ${land.getDensity()}`;
    document.body.appendChild(info);
});

function domeneTilLand(domenenavn) {
    let x = land.find(land => land.toppdomene === domenenavn);
    return x ? x.navn : "Fant ikke landet";
}

console.log(domeneTilLand(".no")); // Norge
```

2018 - 2

Index

```
class ScoreCounter extends Component {

  scores: Score[] = [];

  render() {
    return(
      <>
      <Card title={'ScoreBoard'}>
        <Row>
          <Column><b>Spiller</b></Column>
          <Column><b>Poeng</b></Column>
          <Column></Column>
        </Row>
      { this.scores.map((score) => (
        <>
          <Row key={score.id}>
            <Column>{score.name}</Column>
            <Column>{score.score}</Column>
            <Column><Button.Success onClick={() => this.updateScore(score.id)}>
                      +
                    </Button.Success>
            </Column>
          </Row>
          <br />
        </>
      ))}
      <Button.Light onClick={() => this.reset()}>Nullstill</Button.Light>
      </Card>
      </>
    )
  }

  updateScore(id: number) {
    scoreService.updateScore((this.scores.find((score) => (score.id === id))).score+1,id)
    .then(() => {this.mounted()})
    .catch((error) => {console.error(error)});
  }

  reset(){
    scoreService.resetScore()
    .then(() => {this.mounted()})
    .catch((error) => {console.error(error)});
  }

  mounted() {
    scoreService.getScores()
    .then((scores) => {this.scores = scores; this.forceUpdate()})
    .catch((error) => {console.error(error)});
  }
}

ReactDOM.render(
  <div>
    <Alert />
    <HashRouter>
      <div>
        <Route exact path="/" render={() => <ScoreCounter />} />
      </div>
    </HashRouter>
  </div>,
  document.getElementById('root')
);
```

services

```
import { pool } from './mysql-pool';

export class Score {
  id: number = 0;
  name: string = '';
  score: number = 0;
}

class ScoreService {

  getScores(){
    return new Promise<Score[]>((resolve, reject) => {
      pool.query('SELECT * FROM Scores', (err, rows) => {
        if(err) reject(err);
        else resolve(rows);
      });
    });
  }

  updateScore(score: number, id: number){
    return new Promise<void>((resolve, reject) => {
      pool.query('UPDATE Scores SET score = ? WHERE id = ?', [score, id], (err, rows) =>
     {
        if(err) reject(err);
        else resolve();
      });
    });
  }

  resetScore(){
    return new Promise<void>((resolve, reject) => {
      pool.query('UPDATE Scores SET score = 0', (err, rows) => {
        if(err) reject(err);
        else resolve();
      });
    });
  }

}
export let scoreService = new ScoreService();
```

2019 - 2

```
class ShoppingList extends Component {

  list: List[] = [];

  render() {
    return (
      <>
      <Card title={'Shopping List'}>
        <Row>
          <Column>Varenavn</Column>
          <Column>Antall</Column>
          <Column>Plukket opp</Column>
          {
            <Column>Slett</Column>
          }
        </Row>
        {
          this.list.map(list =>

              <Row key={list.id}>
                <Column>
                  {list.name}
                </Column>
                <Column>
                  {list.count}
                </Column>
                <Column>
                  {list.collected ? 'Ja' : <Button.Success onClick={() =>
                    this.collect(list.id)}>Plukk opp</Button.Success>}
                </Column>
                { list.collected ?
                  <Column>
                    <Button.Danger onClick={() => this.delete(list.id)}>X</Button.Danger>
                  </Column> : <Column></Column>
                }
              </Row>
          )
          }
          </Card>
          <br />
          <NewItem />
      </>
    );
  }

  delete(id: number) {
    shoppingService.deleteItem(id)
    .then(() => history.go(0));
  }

  collect(id: number) {
    shoppingService.collectItem(id)
    .then(() => history.go(0));
  }

  mounted() {
    shoppingService.getList().then(list => this.list = list);
  }
}

class NewItem extends Component {
  list = new List();

  render() {
    return (
      <>
        <Card title={'Legg til ny vare'}>
              <Form.Label>Navn:</Form.Label>
              <Form.Input type="text" value={this.list.name} onChange={(e) =>
                          this.list.name = e.target.value} />
              <Form.Label>Antall:</Form.Label>
              <Form.Input type="number" value={this.list.count} onChange={(e) =>
                          this.list.count = parseInt(e.target.value)} />
              <br />
            <Button.Success onClick={() => this.add(this.list)}>Legg til</Button.Success>
        </Card>
      </>
    );
  }

  add(x: List) {
    shoppingService.addItem(x).then(() => {
      history.go(0);
    });
  }
}

ReactDOM.render(
  <div>
    <Alert />
    <HashRouter>
      <div>
        <Route exact path="/" component={ShoppingList} />
      </div>
    </HashRouter>
  </div>,
  document.getElementById('root')
);


//services
import { pool } from './mysql-pool';

export class List {
  id: number = 0;
  name: string = '';
  count: number = 0;
  collected: boolean = false;
}

class ShoppingService {

  getList() {
    return new Promise<List[]>((resolve, reject) => {
      pool.query('SELECT * FROM ShoppingList', (err, result) => {
        if (err) reject(err);
        else resolve(result);
      });
    });
  }

  addItem(x: List) {
    return new Promise<List>((resolve, reject) => {
      pool.query('INSERT INTO ShoppingList (name,count,collected) VALUES (?,?,?)',
      [x.name, x.count, false], (err, result) => {
        if (err) return reject(err);
        resolve(result);
      });
    });
  }

  collectItem(id: number) {
    return new Promise<number>((resolve, reject) => {
      pool.query('UPDATE ShoppingList SET collected=? WHERE id=?', [true, id],
      (err, result) => {
        if (err) return reject(err);
        resolve(result);
      });
    });
  }

  deleteItem(id: number) {
    return new Promise<number>((resolve, reject) => {
      pool.query('DELETE FROM ShoppingList WHERE id=?', [id], (err, result) => {
        if (err) return reject(err);
        resolve(result);
      });
    });
  }

}

export let shoppingService = new ShoppingService();
```

2020 - 1

```
class Circle {
    constructor(radius) {
        (radius <= 0 || radius === undefined || isNaN(radius)) ? this.radius = 1 :
          this.radius = radius;
        this.color = 'red'
    }
    area() {
        return (Math.PI * this.radius * this.radius).toFixed(2);
    }
    circumference() {
        return (2 * Math.PI * this.radius).toFixed(2);
    }
    rad() {
        return (this.radius).toFixed(2);
    }
}

// Example Circles

let exampleBtn = document.createElement("button");
    exampleBtn.innerHTML = "Create Example Circles";
document.body.appendChild(exampleBtn);

let exampleInfo = document.createElement("p");
    exampleInfo.id = "exampleInfo";
document.body.appendChild(exampleInfo);

exampleBtn.onclick = () => {
    let exampleCircle1 = new Circle();
    let exampleCircle2 = new Circle(2, "blue");

    document.getElementById("exampleInfo").innerText = `Circle 1:
    Radius: ${exampleCircle1.rad()} | Area: ${exampleCircle1.area()} |
    Circumference: ${exampleCircle1.circumference()} | Color: ${exampleCircle1.color}
    ______________________________________________________
    Circle 2:
    Radius: ${exampleCircle2.rad()} | Area: ${exampleCircle2.area()} |
    Circumference: ${exampleCircle2.circumference()} | Color: ${exampleCircle2.color}
    `
}

// Custom Circles

let circleInfo = document.createElement('p');
circleInfo.innerHTML = "";
circleInfo.id = "circleInfo";

let radiusinput = document.createElement('input');
radiusinput.type = "number";
radiusinput.id = "radiusInput";
radiusinput.placeholder = "Enter radius";
document.body.appendChild(radiusinput);
let colorinput = document.createElement('input');
colorinput.type = "text";
colorinput.id = "colorInput";
colorinput.placeholder = "Enter color";
document.body.appendChild(colorinput);


let circleBtn = document.createElement('button');
circleBtn.innerHTML = "Create Circle"
circleBtn.onclick = () => {
    let radius = parseInt(document.getElementById("radiusInput").value);
    let color = document.getElementById("colorInput").value;
    let circle1 = new Circle(radius, color);
    console.log(circle1)
    document.getElementById("circleInfo").innerText += `Radius: ${circle1.rad()} |
    Area: ${circle1.area()} | Circumference: ${circle1.circumference()} |
    Color: ${circle1.color}
    `;
}

document.body.appendChild(circleBtn);
document.body.appendChild(circleInfo);

// Cube
class Cube{
    constructor(circle) {
        let area = circle.area();
        this.side = Math.sqrt(area / 6);
    }
}

let cubeCircle = new Circle(4)
let cube = new Cube(cubeCircle);
console.log(cube);
```

2020 - 2

```
class Piece {
    constructor(x, y, color) {
        // alphabet from a to h
        let alphabet = ["A", "B", "C", "D", "E", "F", "G", "H"];
        this.x = x;
        this.y = y;
        this.position = alphabet[x] + (y+1)
        this.color = color;
    }
}

// Chess pieces
class Pawn extends Piece {
    constructor(x, y, color) {
        super(x, y, color);
        this.type = "pawn";
    }
}
class Rook extends Piece {
    constructor(x, y, color) {
        super(x, y, color);
        this.type = "rook";
    }
}
class Knight extends Piece {
    constructor(x, y, color) {
        super(x, y, color);
        this.type = "knight";
    }
}
class Bishop extends Piece {
    constructor(x, y, color) {
        super(x, y, color);
        this.type = "bishop";
    }
}
class Queen extends Piece {
    constructor(x, y, color) {
        super(x, y, color);
        this.type = "queen";
    }
}
class King extends Piece {
    constructor(x, y, color) {
        super(x, y, color);
        this.type = "king";
    }
}

// create a chess board
let chessBoard = [];
for (let i = 0; i < 8; i++) {
    chessBoard[i] = [];
    for (let j = 0; j < 8; j++) {
        chessBoard[i][j] = null;
    }
}

// create chess pieces starting position
chessBoard[0][0] = new Rook(0, 0, "white");
chessBoard[0][1] = new Knight(1, 0, "white");
chessBoard[0][2] = new Bishop(2, 0, "white");
chessBoard[0][3] = new Queen(3, 0, "white");
chessBoard[0][4] = new King(4, 0, "white");
chessBoard[0][5] = new Bishop(5, 0, "white");
chessBoard[0][6] = new Knight(6, 0, "white");
chessBoard[0][7] = new Rook(7, 0, "white");
for (let i = 0; i < 8; i++) {
    chessBoard[1][i] = new Pawn(i, 1, "white");
}
chessBoard[7][0] = new Rook(0, 7, "black");
chessBoard[7][1] = new Knight(1, 7, "black");
chessBoard[7][2] = new Bishop(2, 7, "black");
chessBoard[7][3] = new Queen(3, 7, "black");
chessBoard[7][4] = new King(4, 7, "black");
chessBoard[7][5] = new Bishop(5, 7, "black");
chessBoard[7][6] = new Knight(6, 7, "black");
chessBoard[7][7] = new Rook(7, 7, "black");
for (let i = 0; i < 8; i++) {
    chessBoard[6][i] = new Pawn(i, 6, "black");
}
console.table(chessBoard)
```

2020 - 3

```
const history = createHashHistory();
// Use history.push(...) to programmatically change path,
//for instance after successfully saving a show

class ShowList extends Component {
  shows: Show[] = [];
  ratings: Rating[] = [];
  search: string = '';

  render() {
    return (
      <>
      <Card title="Barne-TV Programmer">
      <Button.Success onClick={() => history.push('/create')}>
        Legg til program
      </Button.Success>
      <br />
      <Form.Label>Søk</Form.Label>
      <Form.Input type="text" value={this.search} onChange={(event) =>
(this.search = event.target.value)}/>
      <br /><br />
      {this.shows
      .filter((show) => (show.title.toLowerCase().includes(this.search.toLowerCase())))
      .map((show) => (
        <>
            <Card title={show.title} key={show.id}>
              <Row>
                <Column>{show.description}</Column>
                </Row>
              <Row>
                <Column>Terningkast: {''}
                  { this.ratings
                    .filter((showRating) => (showRating.showId === show.id))
                    .reduce((average, showRating, _index, ratings) =>
                    (average + showRating.rating / ratings.length), 0).toFixed(2)
                  }
                </Column>
                <Column>
                  Gi terningkast <br />
                  { [1,2,3,4,5,6].map((rates)=>(
                    <img key={rates}
src={rates+'.png'} width={'60vh'} style={{cursor:'pointer'}} onClick={() =>
this.rate(rates,show.id)}/>
                  ))
                  }
                  {' '}
                </Column>
              </Row>
              <Row>
                <Column>
                  <Button.Light onClick={() =>
history.push('/shows/' + show.id)}>Rediger</Button.Light>
                </Column>
              </Row>
            </Card>
            <br />
            </>
          ))}
      </Card>
      </>
    );
  }

  rate(rating: number,showId: number){
    showService
    .addRating(rating, showId)
    .then(() => {this.mounted()});
  }

  mounted() {
    showService
    .getShows()
    .then((shows) => (this.shows = shows))
    .catch(err => console.error(err));

    showService
    .getRatings()
    .then((rating) => (this.ratings = rating))
    .catch(err => console.error(err));
  }
}

class ShowCreate extends Component {
  show: Show = new Show();

  render(){
    return(
      <>
      <Card title="Legg til program">
          <Form.Label>Tittel</Form.Label>
          <Form.Input type="text" value={this.show.title} onChange={(event) =>
(this.show.title = event.target.value)} />
          <Form.Label>Beskrivelse</Form.Label>
          <Form.Input type="text" value={this.show.description} onChange={(event) =>
(this.show.description = event.target.value)} />
          <br />
          <Row>
            <Column>
              <Button.Success onClick={() => this.add()}>Legg til</Button.Success>
              {' '}
              <Button.Danger onClick={() => history.push('/')}>Avbryt</Button.Danger>
            </Column>
          </Row>
      </Card>

      </>
    )
  }

  add() {
    showService
    .createShow(this.show)
    .then(() => history.push('/'))
    .catch(err => console.error(err));
  }
}

class ShowEdit extends Component<{ match: { params: { id: number } } }> {
  show: Show = new Show();

  render(){
    return(
      <>
      <Card title="Rediger program">
          <Form.Label>Tittel</Form.Label>
          <Form.Input type="text" value={this.show.title} onChange={(event) =>
(this.show.title = event.target.value)} />
          <Form.Label>Beskrivelse</Form.Label>
          <Form.Input type="text" value={this.show.description} onChange={(event) =>
(this.show.description = event.target.value)} />
          <br />
          <Row>
            <Column>
              <Button.Success onClick={() => this.edit()}>Lagre</Button.Success>
              {' '}
              <Button.Light onClick={() => history.push('/')}>Avbryt</Button.Light>
              {' '}
              <Button.Danger onClick={() => this.handleDelete()}>Slett</Button.Danger>
            </Column>
          </Row>
      </Card>
      </>
    )
  }
  handleDelete() {
    { confirm('Vil du slette programmet?') ?
      showService.deleteShow(this.show.id)
      .then(() => history.push('/'))
      .catch(err => console.error(err))
     : console.log('cancel');
  }
}

  edit() {
    showService.updateShow(this.show)
    .then(() => history.push('/'))
    .catch(err => console.error(err));
  }
  mounted(){
    showService
    .getShow(this.props.match.params.id)
    .then((show) => (this.show = show))
    .catch(err => console.error(err));
  }
}

ReactDOM.render(
  <div>
    <Alert />
    <HashRouter>
```

```
      <div>
        <Route exact path="/" component={ShowList} />
        <Route exact path="/shows/:id" component={ShowEdit} />
        <Route exact path="/create" component={ShowCreate} />
      </div>
    </HashRouter>
  </div>,
  document.getElementById('root')
);


//services
import { powerMonitor } from 'electron';
import { pool } from './mysql-pool';

export class Show {
  id: number = 0;
  title: string = '';
  description: string = '';
}

export class Rating {
  rating: number = 0;
  showId: number = 0;
}

class ShowService {
  getShows() {
    return new Promise<Show[]>((resolve, reject) => {
    pool.query('SELECT * FROM Shows', (error, results) => {
      if (error) return reject(error);
      resolve(results);
      });
    });
  }

  getShow(id: number) {
    return new Promise<Show>((resolve, reject) => {
    pool.query('SELECT * FROM Shows WHERE id=?', [id], (error, results) => {
      if (error) return reject(error);

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

  updateShow(show: Show) {
    return new Promise((resolve, reject) => {
      pool.query(
        'UPDATE Shows SET title=?, description=? WHERE id=?',
        [show.title, show.description, show.id],
        (error, results) => {
          if (error) return reject(error);
          resolve(results);
        });
    });
  }

  createShow(show: Show) {
    return new Promise<Show>((resolve, reject) => {
    pool.query(
      'INSERT INTO Shows (title, description, id) VALUES (?, ?, ?)',
      [show.title, show.description, show.id],
      (error,results) => {
        if (error) return reject(error);
        resolve(results);
      });
    });
  }

  deleteShow(id: number) {
    return new Promise((resolve, reject) => {
      pool.query('DELETE FROM Shows WHERE id=?', [id], (error,results) => {
        pool.query('DELETE FROM ShowRatings WHERE showId=?', [id], (error,results) => {
          if (error) return reject(error);
          resolve(results);
          });
        if (error) return reject(error);
        resolve(results);
      });
    });
  }

  getRatings() {
    return new Promise<Rating[]>((resolve, reject) => {
    pool.query('SELECT rating, showId FROM ShowRatings', (error, results) => {
      if (error) return reject(error);
      resolve(results);
    })});
  }

  addRating(rating: number, showId: number) {
    return new Promise<void>((resolve, reject) => {
    pool.query(
      'INSERT INTO ShowRatings (rating, showId) VALUES (?, ?)',
      [rating, showId],
      (error) => {
        if (error) return reject(error);

        resolve();
      }
    )});
  }


}
export let showService = new ShowService();
```

2021 - 1

```
class Kjoretoy {
    constructor(makshastighet, kjorelengde) {
        this.makshastighet = makshastighet;
        this.kjorelengde = kjorelengde;
    }
}

let kjoretoy1 = new Kjoretoy(120, 150);

document.body.innerText += `Kjøretøy 1: Makshastighet: ${kjoretoy1.makshastighet} km/t |
Kjørelengde: ${kjoretoy1.kjorelengde} km \n\n`;

class Buss extends Kjoretoy {
    constructor(makshastighet, kjorelengde, makspassasjerer) {
        super(makshastighet, kjorelengde);
        this.makspassasjerer = makspassasjerer;
    }
    sjekkAntall(antallpassasjerer) {
        if (antallpassasjerer > this.makspassasjerer) {
            return false;
        }
        return true;
    }
    leiePris(){

    }
}

let buss1 = new Buss(90, 200, 65);

document.body.innerText += `Buss 1: Makshastighet: ${buss1.makshastighet} km/t |
Kjørelengde: ${buss1.kjorelengde} km | Maks passasjerer: ${buss1.makspassasjerer} \n\n`;

let checkInp = document.createElement('input');
checkInp.type = "number";
checkInp.id = "antallpassasjerer";
checkInp.placeholder = "Enter number of passengers";
document.body.appendChild(checkInp);

let checkBtn = document.createElement("button");
    checkBtn.innerHTML = "Check if buss can hold passengers";
document.body.appendChild(checkBtn);

let checkInfo = document.createElement("p");
    checkInfo.id = "checkInfo";
document.body.appendChild(checkInfo);

    checkBtn.onclick = () => {
        let antallpassasjerer = document.getElementById("antallpassasjerer").value;
        if (buss1.sjekkAntall(antallpassasjerer)) {
            checkInfo.innerText += `Buss 1 can hold ${antallpassasjerer} passengers.
\n\n`;
        }
        else {
            checkInfo.innerText += `Buss 1 cannot hold ${antallpassasjerer} passengers.
\n\n`;
        }
    }
```

2021 - 3

```
class ChatList extends Component {
  chatRooms: ChatRoom[] = [];
  newChatRoom: ChatRoom = new ChatRoom();
  search: string = '';

  render() {
    return(
      <>
        <Card title="Chat Rooms">
          <Form.Label>Søk</Form.Label>
          <Form.Input type="text" value={this.search} onChange={(event) =>
(this.search = event.target.value)}/>
          <br /><br />
          {this.chatRooms
          .filter((chatRoom) => (chatRoom.title.toLowerCase().
includes(this.search.toLowerCase())))
          .map((chatRoom) => (
            <>
              <Card title={chatRoom.title} key={chatRoom.id}>
                <Row>
                  <Column>{chatRoom.description}</Column>
                </Row>
                <Row>
                  <Column>
                    <Button.Light onClick={() =>
history.push('/chat/' + chatRoom.id)}>Gå til chat</Button.Light>
                      {' '}
                    <Button.Danger onClick={()=>
this.delete(chatRoom.id)}>Slett Chat</Button.Danger>
                  </Column>
                </Row>
              </Card>
              <br />
              </>
              ))}
        </Card>
        <br />
        <Card title="Nytt rom">
          <Form.Label>Tittel</Form.Label>
          <Form.Input type="text" value={this.newChatRoom.title} onChange={(event) =>
(this.newChatRoom.title = event.target.value)}/>
          <Form.Label>Beskrivelse</Form.Label>
          <Form.Input type="text" value={this.newChatRoom.description}
onChange={(event) => (this.newChatRoom.description = event.target.value)}/>
          <br />
          <Button.Success onClick={() => this.add()}>Legg til</Button.Success>
        </Card>
      </>
    )
  }

  add(){
    chatService.createChatRoom(this.newChatRoom)
    .then(() => this.mounted())
    .catch(err => console.error(err));
  }

  delete(id: number){
    chatService.deleteChatRoom(id)
    .then(() => this.mounted())
    .catch(err => console.error(err));
  }

  mounted(){
    chatService.getChatRooms()
    .then(chatRooms => this.chatRooms = chatRooms)
    .catch(err => console.error(err));
  }
}

class ChatDetails extends Component<{ match: { params: { id: number } } }> {
  chatRoom: ChatRoom = new ChatRoom();
  messages: ChatMessage[] = [];
  newMessage: ChatMessage = new ChatMessage();

  render() {
    return(
      <>
```

```
        <Card title={this.chatRoom.title}>
          <Row>
            <Column>{this.chatRoom.description}</Column>
            { this.messages.map((messsages) =>
              <Row><Column>{'>'} {messsages.text}</Column></Row>
              )
            }
          </Row>
        </Card>
        <br />
        <Card title="Melding">
          <Form.Input type="text" value={this.newMessage.text} onChange={(event) =>
(this.newMessage.text = event.target.value)}/>
          <br />
          <Button.Success onClick={() => this.addMessage()}>Send</Button.Success>
        </Card>
        <Button.Light onClick={() => history.push('/')}>Tilbake</Button.Light>
      </>
    )
  }

  addMessage(){
    console.log(this.newMessage)
    chatService
    .addMessage(this.newMessage.text, this.chatRoom.id)
    .then(() => this.mounted())
    .catch(err => console.error(err));
  }

  mounted(){
    chatService.getChatRoom(this.props.match.params.id)
    .then(chatRoom => this.chatRoom = chatRoom)
    .catch(err => console.error(err));
    chatService.getMessages(this.props.match.params.id)
    .then(messages => this.messages = messages)
    .catch(err => console.error(err));
  }
}

ReactDOM.render(
  <div>
    <Alert />
    <HashRouter>
      <div>
        <Route exact path="/" component={ChatList} />
        <Route path="/chat/:id" component={ChatDetails} />
      </div>
    </HashRouter>
  </div>,
  document.getElementById('root')
);


//services
import { pool } from './mysql-pool';

export class ChatRoom {
  id: number = 0;
  title: string = '';
  description: string = '';
}

export class ChatMessage {
  text: string = '';
  chatRoomId: number = 0;
}

class ChatService {

  getChatRooms(){
    return new Promise<ChatRoom[]>((resolve, reject) => {
      pool.query('SELECT * FROM ChatRooms', (error, results) => {
        if (error) return reject(error);
        resolve(results);
      });
    });
  }

  getChatRoom(id: number) {
    return new Promise<ChatRoom>((resolve, reject) => {
    pool.query('SELECT * FROM ChatRooms WHERE id=?', [id], (error, results) => {
      if (error) return reject(error);

      resolve(results[0]);
    });
    });
  }
  createChatRoom(chat: ChatRoom) {
    return new Promise<void>((resolve, reject) => {
      pool.query(
        'INSERT INTO ChatRooms (title, description, id) VALUES (?, ?, ?)',
        [chat.title, chat.description, chat.id],
        (error) => {
          if (error) return reject(error);
          resolve();
        });
    });
  }

  deleteChatRoom(id: number) {
    return new Promise((resolve, reject) => {
      pool.query('DELETE FROM ChatRooms WHERE id=?', [id], (error,results) => {
        pool.query('DELETE FROM Messages WHERE chatRoomId=?', [id], (error,results) => {
          if (error) return reject(error);
          resolve(results);
        });
        if (error) return reject(error);
        resolve(results);
      });
    });
  }

  getMessages(chatRoomId: number) {
    return new Promise<ChatMessage[]>((resolve, reject) =>{
      pool.query('SELECT * FROM Messages WHERE chatRoomId=?', [chatRoomId],
(error, results) => {
        if (error) return reject(error);
        resolve(results);
      });
    });
  }

  addMessage(message: string, chatRoomId: number) {
    return new Promise<void>((resolve, reject) => {
      pool.query(
        'INSERT INTO Messages (text, chatRoomId) VALUES (?, ?)',
        [message, chatRoomId],
        (error) => {
          if (error) return reject(error);
          resolve();
        });
    });
  }

}

export let chatService = new ChatService();
```

items og cart

```
// Services
import { pool } from './mysql-pool';
import { crashReporter } from 'electron';

class Item {
  id: number = 0;
  name: string = '';
  description: string = '';
  price: number = 0;
  count: number = 0;
}

class Cart {
  id: number = 0;
  itemId: number = 0;
  itemCount: number = 0;
}

class Items {

  getItems(){
    return new Promise<Item[]>((resolve,reject ) => {
      pool.query('SELECT * FROM Items', (error,results) => {
        if (error) return reject(error);
        resolve(results);
      })
    })
  }

}

class Orders {

  addItem(itemId: number, itemCount: number){
    return new Promise<Cart>((resolve,reject ) => {
      pool.query('INSERT INTO Orders (itemId, itemCount) VALUES (?, ?)',
[itemId,itemCount], (error,results) => {
        if (error) return reject(error);
        resolve(results);
      })
    });
  }

  removeItem(id: number){
    return new Promise<Cart[]>((resolve,reject ) => {
      pool.query('DELETE FROM Orders WHERE itemId=?', [id], (error,results) => {
        if (error) return reject(error);
        resolve(results);
      })
    })
  }

  getOrders(){
    return new Promise<Cart[]>((resolve,reject ) => {
      pool.query('SELECT * FROM Orders', (error,results) => {
        if (error) return reject(error);
        resolve(results);
      })
    })
  }

  emptyOrders() {
    return new Promise<Cart[]>((resolve,reject ) => {
      pool.query('DELETE FROM Orders', (error,results) => {
        if (error) return reject(error);
        resolve(results);
      })
    })
  }
}

let itemService = new Items();
let cartService = new Orders();

// Main code

const history = createHashHistory(); // Use history.push(...) to programmatically change path, for instance after successfully saving

class Menu extends Component {
  render() {
    return (
      <NavBar brand="StudAdm">
        <NavBar.Link to="/items">Items</NavBar.Link>
        <NavBar.Link to="/cart">Shopping Cart</NavBar.Link>
      </NavBar>
    );
  }
}

class Home extends Component {
  render() {
    return <Card title="Welcome to the shop">
      Check out our items and add them to your cart.
      <br />
      <Button.Light onClick={() => history.push('/items')}>Check out items</Button.Light>
    </Card>;
  }
}

class ItemList extends Component {
  items: Item[] = [];
  orders: Cart[] = [];

  render() {
    return(
      <>
        <Card title="Items">
          <Row>
            <Column><b>Item</b></Column>
            <Column><b>Added/Available</b></Column>
            <Column></Column>
          </Row>
          {this.items.map((item => (
            <>
              <Row key={item.id}>
                <Column>
                  <Row>{item.name}</Row>
                  <Row><i>{item.description}</i></Row>
                </Column>
                <Column>
                {this.orders.filter(order => order.itemId === item.id).
reduce((count, current) => count + current.itemCount, 0)}
                /
                {item.count}
                </Column>
                <Column><Button.Success onClick={() => this.add(item.id)}>
Add to cart </Button.Success></Column>
              </Row>
            </>
          )))}
        </Card>
      </>
    )
  }

  add(itemId: number){
    cartService.addItem(itemId, 1)
    .then(() => {this.mounted()})
    .catch(err => console.error(err))
  }

  mounted(){
    cartService.getOrders()
    .then(orders => {this.orders = orders})
    .catch(err => console.error(err))

    itemService.getItems()
    .then(items => {this.items = items})
    .catch(err => console.error(err))
  }
}

class ShoppingCart extends Component {
  cart: Cart[] = [];
  items: Item[] = [];

  render() {
    return(
      <>
        <Card title="Shopping Cart">
          <Row>
            <Column><b>Name</b></Column>
            <Column><b>Price per item</b></Column>
            <Column><b>Count</b></Column>
            <Column><b>Sum</b></Column>
            <Column></Column>
          </Row>
          {this.items.map((item) => (
            <>
            { this.cart.filter(cart => cart.itemId === item.id).reduce
((count, current) => count + current.itemCount, 0) > 0 &&
              <Row key={item.id}>
                <Column>{item.name}</Column>
                <Column>{item.price} kr</Column>
                <Column>{this.cart.filter(cart => cart.itemId === item.id).
reduce((count, cart) => count + cart.itemCount, 0)}</Column>
                <Column>{this.cart.filter(cart => cart.itemId === item.id).
reduce((count, cart) => count + cart.itemCount, 0) * item.price} kr </Column>
                <Column><Button.Light onClick={() => this.remove(item.id)}>Remove
</Button.Light></Column>
              </Row>
            }
            </>
          ))}
          <Row>
            <Column><b>Sum</b></Column>
            <Column></Column>
            <Column></Column>
            <Column><b>{this.cart.reduce((count, cart) => count +
this.items[cart.itemId-1].price, 0)} kr</b></Column>
            <Column></Column>
            {/* {console.log(this.cart.map((cart) => this.items[cart.itemId-1]))} */}
          </Row>
        </Card>
        <Button.Danger onClick={()=>this.empty()}>Empty Cart</Button.Danger>
      </>
    )
  }
  remove(id: number){
    cartService.removeItem(id)
    .then(() => {this.mounted()})
    .catch(err => console.error(err))
  }

  empty(){
    cartService.emptyOrders()
    .then(() => {this.mounted()})
    .catch(err => console.error(err))
  }

  mounted(){
    itemService.getItems()
    .then(items => {this.items = items})
    .catch(err => console.error(err))

    cartService.getOrders()
    .then(orders => {this.cart = orders})
    .catch(err => console.error(err))
  }
}


ReactDOM.render(
  <div>
    <Alert />
    <HashRouter>
      <div>
        <Menu />
        <Route exact path="/" component={Home} />
        <Route exact path="/items" component={ItemList} />
        <Route exact path="/cart" component={ShoppingCart} />
      </div>
    </HashRouter>
  </div>,
  document.getElementById('root')
);
```