# Roast

Filip Sunnemar
Christian Ågren
Fredrik Malmborg

# Upplevelse av backend

- Lätt:

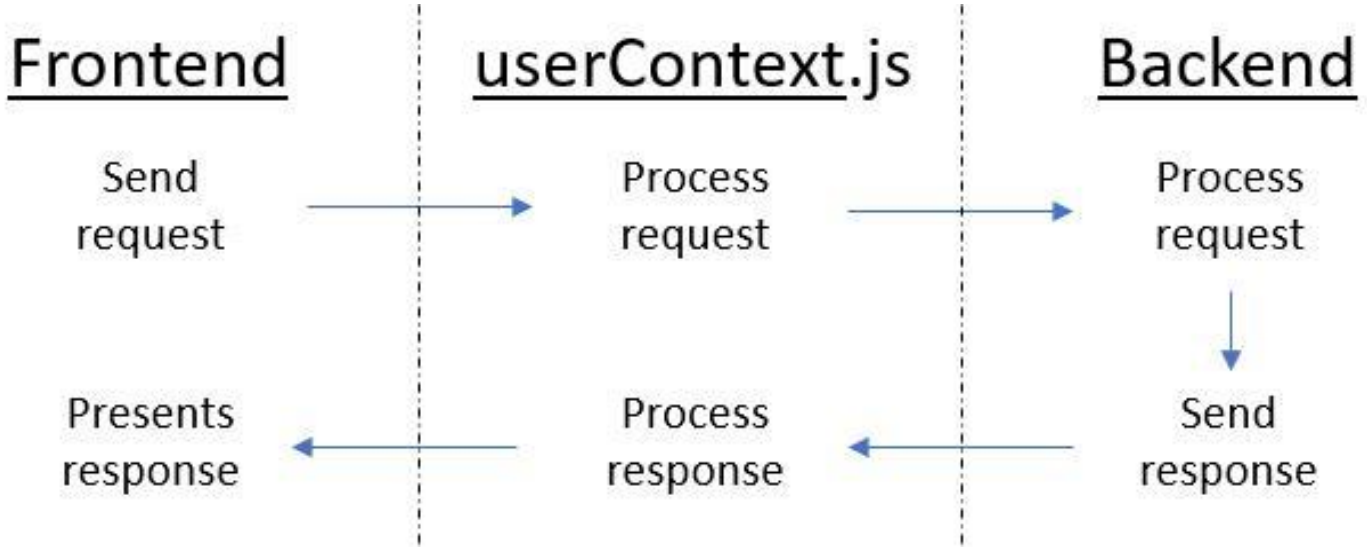
- Express och socket.io var ganska intuitivt
- Logisk struktur

## Svårt/Negativt:

- Ny utmaning med proxy
- Fick ingen visuell feedback när saker inte gjorde vad som var tänkt på backend. Blev svårt att felsöka då.

# Kodens struktur och flöde

- En front-end med react och material ui
- Vi har en user context med react och socket.io
- Vi har backend med socket.io och express



# Server

- Vi har två listor med öppna och stängda rum som vi fyller på med varje rum som skapas.

```
// Connection, servern måste vara igång för att front-end ska fungera,
io.on("connection", function (socket) {
    console.log("made socket connection", socket.id);

    let lockedRooms = [],
        openRooms = [];

    roomInformation.forEach((room) => {
        const availableRoom = {
            name: room.name,
            id: room.id,
            users: room.users,
            password: room.password,
            color: room.color,
        };
        if (room.password.length != 0) {
            lockedRooms.push(availableRoom);
        } else {
            openRooms.push(availableRoom);
        }
    });
```

# UserContext

- En user context där vi initierar och hanterar funktioner och variablers state

```
export const UserContext = React.createContext({
    name: "",
    socket: user.socket,
});

export default class UserProvider extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            name: "",
            socket: user.socket,

            connectedRoom: "",
            connectedRoomColor: "",

            joinRoom: this.joinRoom,
            createName: this.createName,
            availableRooms: {},
            chatlog: [],

            leaveChatRoom: this.leaveChatRoom,
            createNewMessage: this.createNewMessage,
            createNewRoom: this.createNewRoom,

            invalidRequest: this.invalidRequest,

            emitTyping: this.emitTyping,
            usersTyping: [],

            firstTime: true,
        };
```

# Emits i UserContext

- Vi skickade till server när någon klickat på front-end för att till exempel gå in i rum, skapa meddelande eller skapa ett nytt rum.

```
this.state.socket.emit("join room", {
    name,
    roomId,
    prevRoomId,
});
```

```
createNewMessage = (messageValue) => {
    this.state.socket.emit("message", {
        roomId: this.state.connectedRoom,
        name: this.state.name,
        message: messageValue,
    });
};

createNewRoom = (roomValues) => {
    this.state.socket.emit("create room", {
        id: roomValues.roomId,
        password: roomValues.roomPassword,
        color: roomValues.roomColor,
    });
};
```

# Servern lyssnar…

- Servern lyssnar på emitsen och skickar informationen som behövs till relevanta sockets.
- Historiken för rummet finns redan i rummet så servern behöver endast fylla på med det nya meddelandet.

```
socket.on("message", (newMessage) => {
    const { history, color } = roomInformation.find(
        (h) => h.id === newMessage.roomId
    );

    const message = {
        name: newMessage.name,
        message: newMessage.message,
        color: color,
        client: true,
    };

    history.push(message);

    io.to(newMessage.roomId).emit("user message", message);
});
```

# UserContext

- Vi lyssnar på servern i user context för att hantera front-end beroende på vad som skickats från servern.

```
this.state.socket.on("connection successful", (data) =>
    this.setAvailableRoomsInState(data)
);
this.state.socket.on("join successful", (data) =>
    this.setRoomInState(data)
);
this.state.socket.on("chatlog", (data) => this.generateChatLog(data));

this.state.socket.on("user message", (data) =>
    this.generateChatMessage(data)
);
this.state.socket.on("notice", (data) => this.generateChatMessage(data));

this.state.socket.on("created new room", (data) =>
    this.updateAvailableRooms(data)
);

this.state.socket.on("room has been created", (data) => this.joinCreatedRoom(data));

this.state.socket.on("user left room", (data) =>
    this.updateUsersinRoom(data)
);

this.state.socket.on("user joined room", (data) =>
    this.updateUsersinRoom(data)
);
this.state.socket.on("remove room", (data) => this.removeRoom(data));

this.state.socket.on("typing", (data) => this.handleTyping(data));
```

# Styling

- Det blev långa filer så vi valde att bryta ut stylingen i egna js-filer.

# Saker vi kunnat göra annorlunda

- Sätta upp projektet med typescript för att få hjälp när man bygger saker man inte är van vid.
- Server.js och UserContext.js är väldigt stora och bulkiga filer som hade varit bra att bryta ut i mindre komponenter.

# Demo