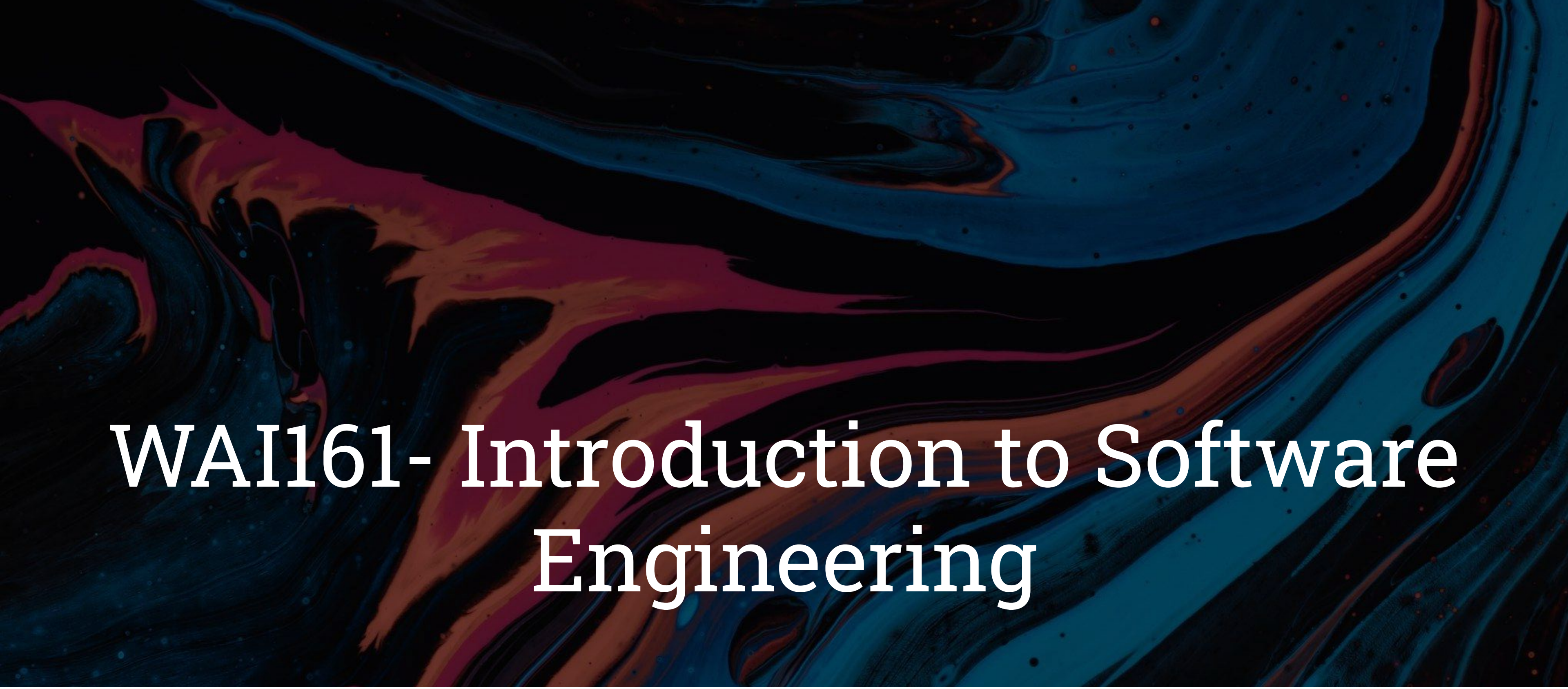
Welcome back...

To get the latest code so that you can follow along, go to `github.com/WarwickAI/wai161` to get the latest code (in Session 2).

Once downloaded, open the folder in VSCode and in the terminal run `npm install` to install the packages

Answer this quiz on content from the previous session whilst we wait



WAI161- Introduction to Software Engineering

Tutorial 3 - Server For User & Message Management + Use Effect Hook

Course Breakdown

Tutorial 0: Setup VSCode, Git and NodeJS (Medium article)

Tutorial 1: Introduction to web development, setting up NextJS project, some UI stuff

Tutorial 2: More React; state management, component lifecycle (+hooks), basic querying

Tutorial 3: Creating and developing server in NodeJS

Tutorial 4: Finishing touches and deployment to Vercel

Where are we?

- Set up a React project
- Added some UI libraries
- Created a basic UI
- Created message functionality
- Used a Hugging Face model to do NLP analysis and send a message in our chat

Today we are going to…

- Create a Web Server using NextJS to manage sending and storing our messages, as well as login and authentication

- Use NextAuth to require user login for sending messages

- Utilise the `useEffect` React hook to handle updating messages

Setting up Login and Authentication

We are going to use NextAuth:

- Gives us utilities to very easily integrate login
- Can use different providers like Google, Apple, Warwick Uni etc.

We will use Discord, here's how to set it up:

1. Create a Discord App (follow this)
2. Put the **`DISCORD_CLINET_ID`** and **`_SECRET`** in the **`.env`** file
3. Set the **`NEXTAUTH_SECRET`** to something random...

Add Login Button

In index.tsx, we want users to be able to sign in and out.

NextAuth provides us with the following hook to access login information:

```
const { data: sessionData } = useSession();
```

Put this somewhere at the top of your page

Add the following button somewhere in your page:

```
<button
    className="btn-sm btn mt-2"
    onClick={sessionData ? () => signOut() : () => signIn()}
>
    {sessionData ? "Sign out of " + sessionData.user?.email : "Sign in"}
</button>
```

Test Login in

Button should display Sign In

After logging in with Discord, the button should display your email.

Try setting the Send Button to disabled unless the user is logged in

First, Let’s Set-Up our Database

We use Prisma to manage our Database schema and interactions

Run the following to initialise the database:

```
npx prisma migrate dev --name "init"
```

Run some command

Migrate Database to be up-to-date

We are in development environment

Name for migration (in this case this is the first/initial migration)

This will add the **`migrations`** folder in the **`prisma`** directory

Add Message Model to Database

We can add more entity types to the database

Look in **`prisma/schema.prisma`** at models that are already there.

Add the Message entity/model with:

Model name

Type of attribute

Extra properties

References to other tables

```
model Message {
    id        String   @id @default(cuid())
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt
    text      String
    user      User?    @relation(fields: [userId], references: [id])
    userId    String
    ai        Boolean  @default(false)
}
```

We also need to add the relation to User:

```
model User {
    id            String    @id @default(cuid())
    name          String?
    email         String?   @unique
    emailVerified DateTime?
    image         String?
    accounts      Account[]
    sessions      Session[]
    Message       Message[]
}
```

Add this

Now run a migration to *'lock in'* our changes:

```
npx prisma migrate dev --name "add messages"
```

NextJS Server - TRPC

TRPC allows us to create actions that run on the server

Instead of a normal API with endpoints, with TRPC we can use these actions like function.

What do we want a server to manage?

- Manage and store messages
- Handle users

NextJS Server Set-Up

All stored in **`src/server`**

- `common` — Some configuration
- `db` — Some configuration
- `trpc`
  - `router`
    - `_app.ts` — Route configuration (maps names to functions)
    - `auth.ts` — Some authentication routes
    - `example.ts` — Some example routes
  - `context.ts` — Gives us access to user information in routes
  - `trpc.ts` — Sets up procedures for public and private

Add Router to Handle Messages

1. Create a new file in **`trpc/router`** called **`message.ts`**

2. Add the following:

```
import { publicProcedure, router } from "../trpc";

export const messageRouter = router({
    getAll: publicProcedure.query(({ ctx }) => {
        return ctx.prisma.message.findMany();
    }),
});
```

Router name

Specific TRPC function
(in this case returns all the messages from Prisma)

Add Create Message Mutation in Router

We also want to be able to create messages.

Add the following as another TRPC ‘*function*’:

Requires being logged in, therefore protected

```
create: protectedProcedure.input(z.string()).mutation(({ ctx, input }) => {
    // To-Do: Send AI message here also
    return ctx.prisma.message.create({
        data: {
            text: input,
            user: {
                connect: {
                    id: ctx.session.user.id,
                },
            },
        },
    });
}),
```

Mutation instead of query
(since changes some state)

Create new message with inputted text,
and set user to the signed in user

Add New Message Router

We need to tell TRPC about our new router

Therefore, in **`trpc/router/_app.ts`** add:

```
export const appRouter = router({
    example: exampleRouter,
    auth: authRouter,
    message: messageRouter
});
```

Add this

Using Router in Chat App

We can now use these new actions instead of storing the data on client side.

Here's the steps that we need to do to complete this

1. Use TRPC's hooks for fetching and creating messages:

```
const messagesData = trpc.message.getAll.useQuery();

const createMessage = trpc.message.create.useMutation();
```

2. Update Send Button to use **`onClick={() => createMessage.mutate(msg)}`**

3. Update Message component to use new Prisma Message type

Querying Hugging Face on Server

It’s safer and more intuitive to do the call to the NLP model on the server

Therefore let’s move our complex logic for querying the API into our **`create`** message TRPC function

something like this...

```
create: protectedProcedure
    .input(z.string())
    .mutation(async ({ ctx, input }) => {
        const newMsg = await ctx.prisma.message.create({
            data: {
                text: input,
                user: {
                    connect: {
                        id: ctx.session.user.id,
                    },
                },
            },
        });

        const requestInit: RequestInit = {
            method: "POST",
            headers: HEADERS,
            body: JSON.stringify({ inputs: input }),
        };

        const response = await fetch(API_URL, requestInit);
        const json = await response.json();
        console.log(json);

        const emotion = json[0]["generated_text"];

        const newAiMsg = await ctx.prisma.message.create({
            data: {
                text: "Message sounds like " + emotion,
                ai: true,
            },
        });

        return [newMsg, newAiMsg];
    }),
});
```

Using Server for Messages on Client Side

In our UI, we now want to use the server to get the messages instead of storing them locally.

To do this we will still need to keep our messages state, but update this from the result of the server.

We want to get an array of messages as soon as we open the UI, but how would we do this?

`useEffect`

React useEffect hook

First argument is another function

useEffect is a function that takes 2 arguments

```
useEffect(() => {
  ...
}, [...])
```

Second argument is an array of dependencies

What does this do?

When any dependency changes, the function specified will be run

Essentially, useEffect allows us to listen to changes in states and properties and do something

Adding Chat Refresh

Ideally you would solve this using **`sockets`** (e.g. SocketIO).

Instead we are going to solve it by fetching the messages from the server on a regular basis.

How do we setup an action to run at a certain interval?

JavaScript **`setInterval`**

JavaScript Set Interval

To create:

```
const intervalVariable = setInterval(() => {
    ...
}, 500);
```

Function to run at each iteration goes here

Time interval (in ms) between each iteration

To remove/stop:

```
clearInterval(intervalVariable);
```

Adding Message Refresh Interval

We want to create this interval as soon as we load the page, how could we do this?

`useEffect`

So, we can do this as follows:

```
useEffect(() => {
    const messagesRefreshInterval = setInterval(() => {
        // Code to get messages from server using axios...
    }, 500);
}, []);
```

Replace this with the code we had in our other **`useEffect`** hook.

Removing the Interval

But what happens when we close the page? We need to remove the interval.

The function we define in our `useEffect` hook can return **another function**.

This function we return will be run when the component is unloaded/closed.

Removing the Interval

For example, here is how we would do it with the message refresh interval:

```
useEffect(() => {
    const messagesRefreshInterval = setInterval(() => {
        // Code to get messages from server using axios...
    }, 500);
    return () => {
        clearInterval(messagesRefreshInterval);
    };
}, []);
```

Clearing the interval so that it's no longer being used

Safely clearing the interval is quite important

Before next week complete the following:

- Tutorial 3 (slides will be online)

- Create a web server
    - Manages storing messages
    - Function that sends back the messages
    - Function that allows creating a new message
    - Handles NLP analysis using Hugging Face
- Modify the UI to use this server

Next week we will be:

- Adding some finishing touches to the App
- Deploying our UI and Server into production (i.e. accessible online)
    - We will likely be using Heroku and Vercel for this