WAI161- Introduction to Software Engineering

Tutorial 1 - Intro to Web Dev, React + NextJS

Welcome…

Course Overview

Introduction to applied software engineering, specifically web development.

Less focus on theory e.g. software development methodologies.

Developing your own web application and server

Course Breakdown

4 x 2-hour sessions

In-person **recommended** (tutorials uploaded online)

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

Following the Tutorial

- You are welcome to follow along with the live coding.

- Head to *github.com/WarwickAI/wai161* for resources after the session.

- You can work on the content covered after the live tutorial as well as in your own time.

NLP: Natural Language Processing

You will be developing a…

Chatbot that responds to messages with some **NLP** analysis

e.g. Responds with the sentiment (positive/negative)

“I love this product!!!” → Positive: 0.95, Negative: 0.05

It’s up to **you** what analysis you want to do (more on this next week)

Our “Stack” - T3 using `create-t3-app`

A “Stack” in software development is the frameworks/languages used.

node JS

Environment and Engine

<u>Frameworks/Libraries</u>

NEXT.JS — Web app framework using React

tailwindcss — CSS utilities for easy styling

Prisma — Maps a schema to a database and code

and a few others…

TypeScript language

What is React?

- Declarative framework/library using JavaScript (or Typescript) for **interactive user interfaces**
- Developed by Meta
- #1 in it’s area [1]
- Used by most tech companies (Netflix, Facebook, Airbnb…)

[1]: *https://www.better.dev/react-popularity*

Why React?

- Previous option was to use either pure **HTML**, **CSS** and **JS** or **jQuery**.
- Defining UI felt detached from creating the UI interactivity.

- **React** (and other frameworks) solve this by providing the functionality of **HTML**, **CSS** and **JS** all under one roof, plus many more benefits.

```
// In Javascript file
$(document).ready(function(){
      $('.slides_item').css('background','red')
});
// In HTML file
<div class=”slides_item”></div>
<script src="path_to_your_js/file.js"></script>
```

Creating a div with red background using jQuery

$\longrightarrow$

```
function App() {
      return <div style={{backgroundColor: ‘red'}}/>
}
```

Creating a div with red background using React

NextJS

Wraps React with extra functionality and tools:

- Server and static rendering
- Image optimisation
- Easy routing (like `example.com/page1` → renders `PAGE1`)

Let's Create our Project

(make sure you have completed Tutorial 0 prior to these steps)

Open Terminal (Linux/macOS) or CMD (Windows)

Navigate to folder to create project in (e.g. your GitHub folder)

Run:

```
npx create-t3-app@latest
```

NodeJS *execute* command

command to run

use the current version

Follow the prompts, using the default settings (include all packages) and name `wai161`

This will create a folder called `wai161`

Open this folder in VSCode

Project Structure

📁 `node_modules`	Stores external libraries or packages
📁 `prisma`	Database schema
📁 `public`	Anything needed by the user (e.g. images)
📁 `src`	Code source files, you'll spend most of your time here
📄 `tsconfig.json`	TypeScript configuration
📄 `README.md`	Information about project
📄 `package.json`	Project configuration

Run **`npm run dev`** to see the initial App

(make sure you are in the project when running this command)

`src/pages/index.tsx` (`example.com`)

Replace with the following code

```
import { NextPage } from "next";

const Home: NextPage = () => {
  return (
    <div
      className="
        container mx-auto flex min-h-screen flex-col
        items-center justify-center p-4
      "
    >
      <h1 className="
        text-5xl font-extrabold leading-normal
        text-gray-700 md:text-[5rem]
      ">
        Welcome to WAI161
      </h1>

      <p className="mt-4 text-2xl text-gray-600">
        A Warwick AI course creating a web app
      </p>
    </div>
  );
};


export default Home;
```

Imports

Home page component, will be rendered when you go to **`example.com`**

All “components” in React are a function that return JSX (or TSX)

Component Declaration Breakdown

Component “tag” - refers to function name of component OR HTML element

```
<a
    className="App-link"
    href="https://reactjs.org"
    target="_blank"
    rel="noopener noreferrer"
>
    Learn React
</a>
```

Component “props” - properties of this component

Component “children” - list of components to render inside the component

```
<img src={logo} className="App-logo" alt="logo" />
```

Try making some changes

React should hot-reload the page as you make changes

1. Modify the text in the paragraph (`<p>` tag).
2. Add a link (`<a>` tag) to navigate to the warwick.ai website.
3. Add a section (`<div>` tag) with some text that when you click on it it prints some text to the console. You will need to open developer tools in your browser to see this.

Tailwind CSS

Provides classes for quick component styling

text-sm →

```
.text-sm {
   font-size: 0.875rem /* 14px */;
   line-height: 1.25rem /* 20px */;
}
```

bg-slate-800 →

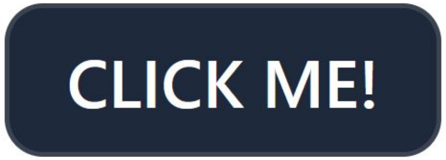
```
.bg-slate-800 {
   --tw-bg-opacity: 1;
   background-color: rgb(30 41 59 / var(--tw-bg-opacity));
}
```

```
<p className="bg-slate-800 text-sm text-white">Hello There</p>
```

DaisyUI

Adds more classes using Tailwind's to create basic components

```
<button className="btn btn-sm bg-slate-800 text-sm text-white">
    Click Me!
</button>
```



We need to install DaisyUI as a package

Adding Packages and Libraries

- **Package**: reusable bits of code
- **Library**: collection of packages

- Over 1 million packages available for JavaScript (or TypeScript) [2]
- Usually get packages from the npm online repository (npmjs.com)

- npm “modules” can be small **utility functions**, full JavaScript **frameworks** or anything in between

[2]: *https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn/*

How to Install a Package

Run:

```
npm install <package-name>
```

Node Package Manager (installed with NodeJS)

Package name

This will add the package to your node_modules folder

and add the package as a dependency in package.json

Adding a Component Library - DaisyUI

- Instead of creating all the components ourselves, use **predefined** ones.
- We can still **modify** these components and **create new** ones (more on this later).
- DaisyUI **one** option, many out there (e.g. ChakraUI , MUI ...).

To install DaisyUI, run this command (or follow daisyui.com/docs/install/)

```
npm install daisyui
```

Add DaisyUI as Tailwind Plugin

Need to let Tailwind know that we can now also use DaisyUI class names.

To do this, modify the **`tailwind.config.cjs`** file to match the following:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{js,ts,jsx,tsx}"],
  theme: {
    extend: {},
  },
  plugins: [require("daisyui")],
  daisyui: {
    themes: false,
  },
};
```

Let's create a simple UI with DaisyUI

Search through DaisyUI's documentation to find out what components they have

1. Add a Text Box Input for entering messages.
2. Add a Button for sending messages.
3. Create a few Message Bubbles.

Now for some interactivity:

1. When the button is clicked, print to the console.
2. When the text in the Text Box Input changes, print it in the console.

Creating Our Own Components

- As well as using components from libraries, we can also create our own.

- For example, we could create a Message component that handles displaying a message.

- This will make our code much more maintainable.

Creating Message Component

1. Create a new **file + folder** → `src/components/Message.tsx`

2. Copy the following code:

```
const Message = () => {
      return (
            … your message bubble components
         );
      }

export default Message;
```

Replace this with the components you were using for your message bubbles

Creating Message Component

3. Add the line `import Message from "./Message";` to `src/pages/index.tsx` at the top.

4. Replace the message bubbles with `<Message/>` in `src/pages/index.tsx`.

Your message bubbles should appear visually the same, just with a much cleaner and reusable method.

We will look at how you can use **properties** and **states** next week to customise components.

Finally, Using Git and GitHub to Track our Code

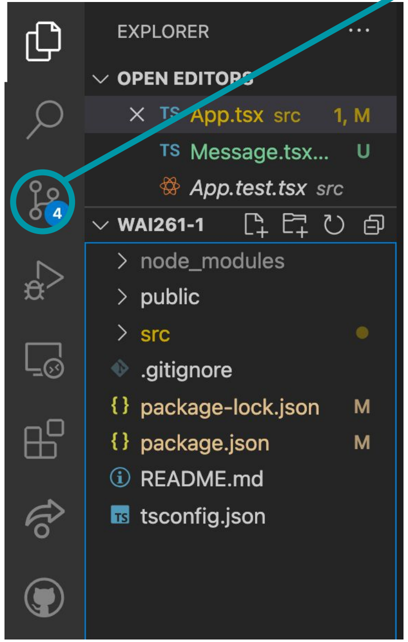
- Your code is currently saved on your device with no version control tracking changes, this has a few problems:
    - Cannot revert to previous versions of your code.
    - Hard to share your code with others.
    - Potential for losing your code by accident or hardware failure.

Therefore, we will be using Git to track our code and GitHub to keep it stored online.

A Git repository was created for us when we created the React project

Committing Our Changes

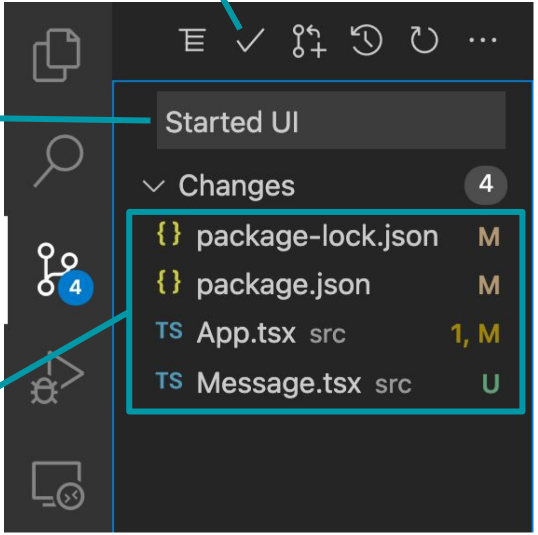
1. Click on the Source Control tab



2. Type a commit message describing the changes you have made

3. Verify the changes you are committing (you may need to add them to the commit)

4. Finally, commit the changes



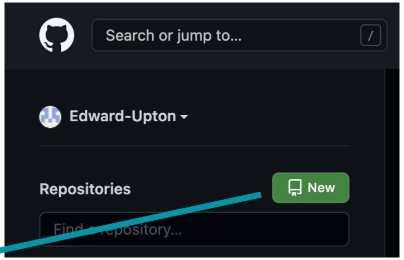
Pushing our Changes to GitHub

- What you just did was “**commit**” the changes to our <u>local</u> repository.
- Now we want to “**push**” (aka upload) these changes to **GitHub**.

- You can imagine **GitHub** as cloud storage for **Git** repositories (there are other options like **BitBucket**/**GitLab**).

- For this next section make sure you have a **GitHub** account.

Creating a Repository on GitHub



1. Navigate to *github.com* and login.
2. Click the "New" button, this will start the process of creating a repository on GitHub.
3. Give the repository a suitable name, I like to name mine the same as my local repository e.g. `wai161`.
4. Click the "Create Repository" button.

We now have a repository setup on GitHub under our account.

Next we need to **connect the two repositories** together.

Adding “Remote” Repository and Pushing Changes

To connect these two repositories, we add the **GitHub** one as a “remote”.

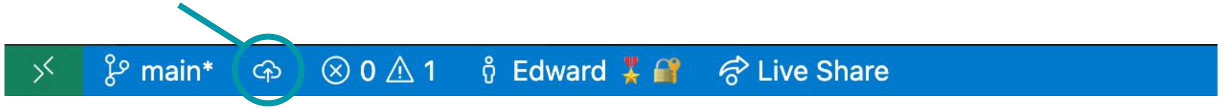
To do this, run the command:

```
git remote add origin https://github.com/<GitHub-username>/<repo-name>.git
```

Then run the command:

```
git branch -m main
```

Now click this button to “push” your changes to **GitHub** (may look different)



Git → GitHub - Problems

You will likely have problems with the last step, the reason being is that you need to **authenticate** yourself since you are trying to commit to a **GitHub** repository.

One solution is to install **GitHub's CLI** (*cli.github.com*), then run:

```
gh auth login
```

If you are still having issues, check out this page:

https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Your Turn

Work through what has been covered here

Create a simple UI, explore different DaisyUI components

Try customising these components via their properties

Before next week complete the following:

- Tutorial 0 and Tutorial 1
- Create a simple chat UI including:
  - Message bubbles
  - Textbox to enter message
  - Button to send message

Next week we will be:

- Customising and modifying custom components using state and the component lifecycle.
- Querying a NLP model to respond to our messages.