Advanced Web Programming

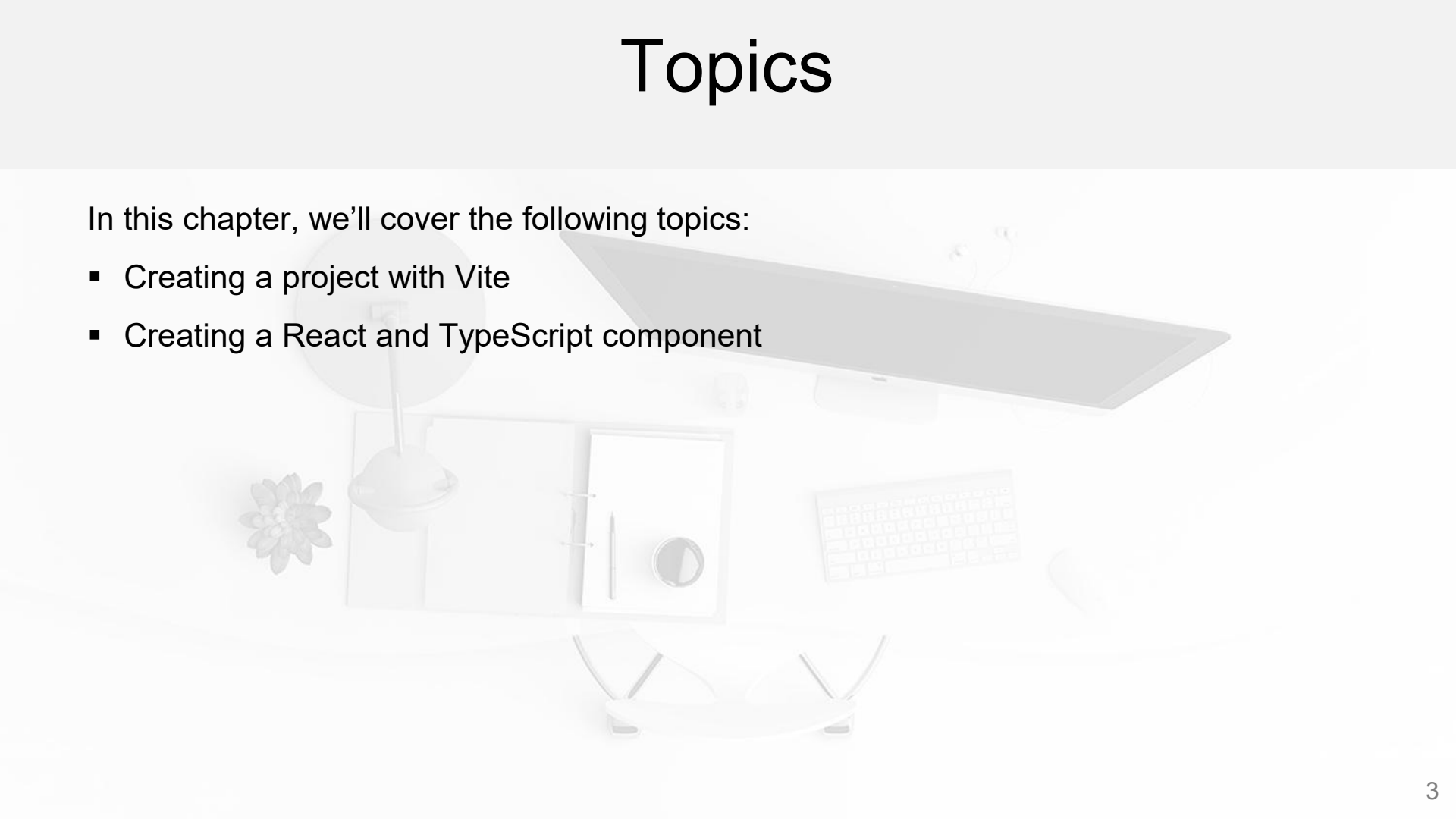
Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

Setting Up
React and TypeScript

Topics

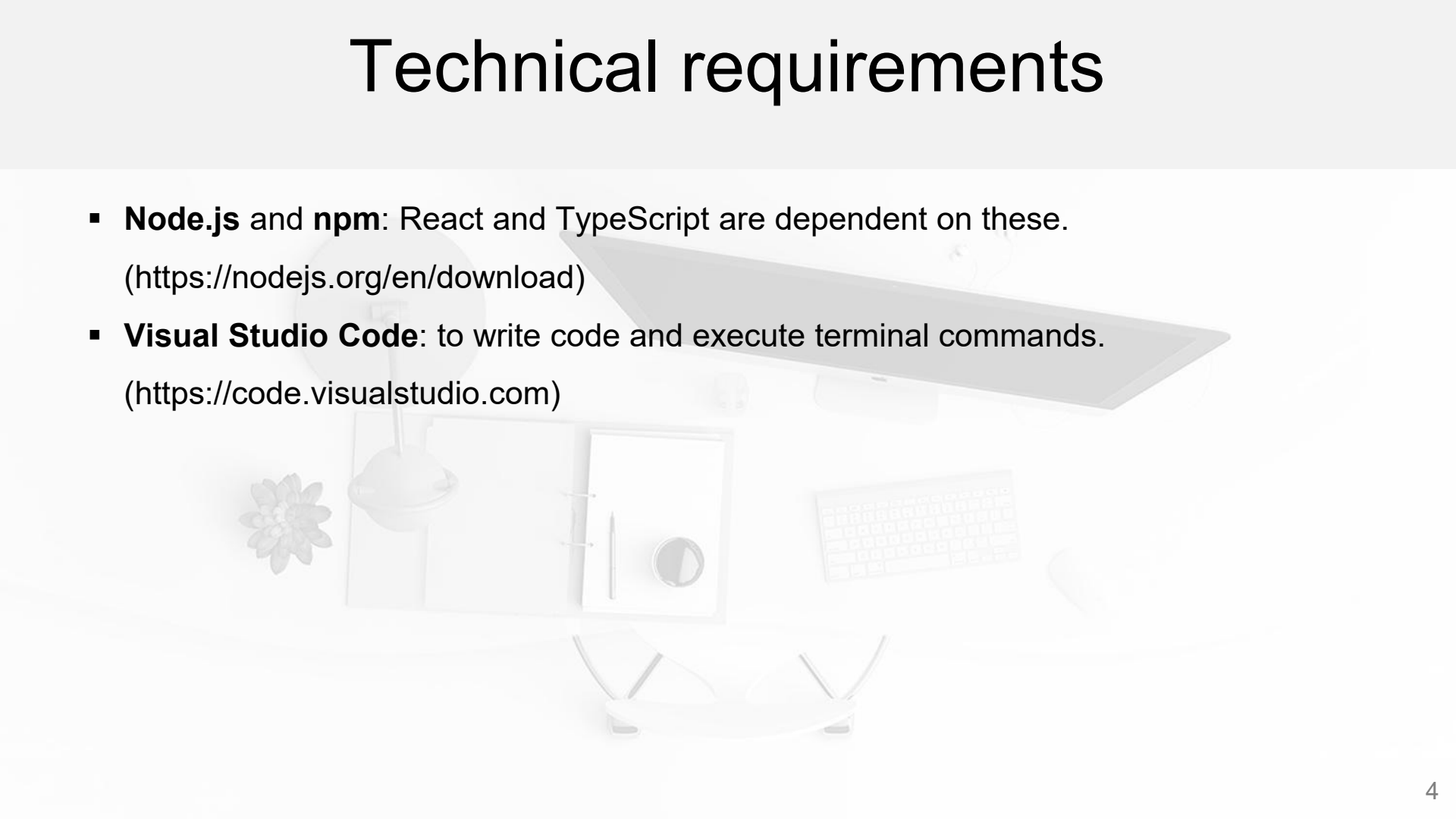
In this chapter, we'll cover the following topics:

- Creating a project with Vite
- Creating a React and TypeScript component



Technical requirements

- **Node.js** and **npm**: React and TypeScript are dependent on these. (https://nodejs.org/en/download)
- **Visual Studio Code**: to write code and execute terminal commands. (https://code.visualstudio.com)



Term

- A **package manager**, such as Yarn or npm. It lets you take advantage of a vast **ecosystem of third-party packages**, and easily install or update them.
- A **bundler**, such as Vite, webpack or Parcel. It lets you **write modular code and bundle it together into small packages** to optimize load time.
- A **compiler** such as Babel. It lets you write modern JavaScript code that still works in older browsers.

Introducing webpack

- Webpack is a tool that **bundles JavaScript source code files together**. It can also bundle CSS and images. It can run other tools such as Babel to transpile React and the TypeScript type checker as it scans the files.

- Webpack is incredibly flexible but, unfortunately, it requires a lot of configuration.

- It is important to understand that webpack isn't a project creation tool. Instead, webpack brings tools such as React and TypeScript together once installed and configured.

Creating a project with Vite

Using Vite

- To use Vite, open Visual Studio Code in a blank folder and run the following command:

  npm create vite@latest app-name -- --template react-ts

  - app-name: is your app name.
  - -- --template react-ts: specifies that template should be used to create the project.

Creating a project with Vite

Adding linting to Visual Studio Code

- **Linting** is the process of checking code for potential problems.
- **ESLint** is a popular tool that can lint React and TypeScript code.
- Vite has already installed and configured ESLint in our project.
- Editors such as Visual Studio Code can be integrated with ESLint to highlight potential problems.

Creating a project with Vite

Adding linting to Visual Studio Code

- Carry out the following steps to **install an ESLint extension into Visual Studio Code**:
  - Open up the Extensions area in Visual Studio Code (File menu > Preference > Extension).
  - Enter ***eslint*** into the extensions list search box. An extension by **Microsoft** called **ESLint** should appear at the top of the list.
  - Click the **Install** button to install the extension.

Creating a project with Vite

Adding linting to Visual Studio Code

- Carry out the following steps to **install an ESLint extension into Visual Studio Code**:
  - Configure the ESLint extension to check React and TypeScript (File menu > Preference > Settings).
  - In the settings search box, enter ***eslint: probe*** and select the **Workspace** tab.
  - Make sure that ***typescript*** and ***typescriptreact*** are on the list. If not, add them using the Add Item button.

Creating a project with Vite

Adding code formatting

- Automatic code formatting ensures code is consistently formatted, which helps its readability.
- Having consistently formatted code also helps developers see the important changes in a code review.
- **Prettier** is a popular tool capable of formatting React and TypeScript code.

Creating a project with Vite

Adding code formatting

- Carry out the following steps to **install and configure Prettier** in the project:
  - Install Prettier using the following command in the terminal in Visual Studio Code:

    npm i -D prettier

  - Prettier has overlapping style rules with ESLint, so install the following two libraries to allow Prettier to take responsibility for the styling rules from ESLint:

    npm i -D eslint-config-prettier eslint-plugin-prettier

Creating a project with Vite

Adding code formatting

- Carry out the following steps to install and configure Prettier in the project:
  - Prettier can be configured in a file called **.prettierrc.json**. Create this file with the following content in the root folder:

```
{
  "printWidth": 100,
  "singleQuote": true,
  "semi": true,
  "tabWidth": 2,
  "trailingComma": "all",
  "endOfLine": "auto"
}
```

We have specified the following:

- Lines wrap at 100 characters
- String qualifiers are single quotes
- Semicolons are placed at the end of statements
- The indentation level is two spaces
- A trailing comma is added to multi-line arrays and objects
- Existing line endings are maintained

Creating a project with Vite

Adding code formatting

- Prettier is now installed and configured in the project.
- Visual Studio Code can integrate with Prettier to automatically format code when source files are saved.
- Let's install a Prettier extension into Visual Studio Code:
  - Open the **Extensions** area in Visual Studio Code and enter ***prettier*** into the extensions list search box. An extension called **Prettier – Code formatter** should appear at the top of the list.
  - Click the **Install** button to install the extension.

Creating a project with Vite

Adding code formatting

- Let’s install a Prettier extension into Visual Studio Code:
  - Open the **Settings** area in Visual Studio Code. Select the **Workspace** tab and make sure the **Format On Save** option is ticked.
  - Click the **Workspace** tab and make sure **Default Formatter** is set to **Prettier - Code formatter.**

Creating a project with Vite

Starting the app in development mode

- Carry out the following steps to start the app **in development mode**:
  - Vite has already created an **npm** script called **run dev**, which runs the app in development mode. Run this script in the terminal as follows:

    npm run dev
  - Open **App.tsx** and change something. After the file is saved, the running app is automatically refreshed.
  - Add a used variable and pass an invalid prop in a JSX element.
  - The problems are caught and reported in Visual Studio Code. The issues are also reported in the browser.
  - Remove the invalid code and stop the app from running before continuing. The shortcut key for stopping the app is **Ctrl + C**

Creating a project with Vite

Producing a production build

- Carry out the following steps to produce a build of the app that can be deployed into production:
  - Vite has already created an npm script called build that produces all the artifacts for deployment to production. Run this script in the terminal as follows:

    npm run build

  - Open the **build** folder – it contains many files. The root file is index.html, which references the other JavaScript, CSS, and image files. All the files are optimized for production with whitespace removed and the JavaScript minified.

Creating a project with Vite

Recap

- The Vite tool can execute the Vite library, specifying the typescript template to create a React and TypeScript project.
- Vite sets up many useful project features, such as linting, CSS support, and SVG support.
- Vite also sets up npm scripts to run the app in development mode and produce a production build.

Creating a React and TypeScript component

Adding a props type

- Follow the steps below:
  - Create a new file in the **src** folder called **Alert.tsx**. Paste in the JavaScript version of the alert component.
  - Add the following type just above the component. This will be the type for the component props:

```
type Props = {
  type?: string;
  heading: string;
  children: React.ReactNode;
  closable?: boolean;
  onClose?: () => void;
};
```

Creating a React and TypeScript component

Adding a props type

- Follow the steps below:
  - Assign the **Props** type to the alert component after the destructed parameters:

```
export function Alert({
  type = "information",
  heading,
  children,
  closable,
  onClose,
}: Props) {
  ...
}
```

Creating a React and TypeScript component

Adding a props type

- Follow the steps below:
  - Open **App.tsx** and replace the header element with the alert component. Don’t forget to import the alert component before using it in the JSX. Don’t pass any props into Alert to test the type checking.
  - Pass in a header prop to Alert, and give it some content.
  - Start the app in development mode if not already running (npm run dev)

Creating a React and TypeScript component

Adding a state type

- Follow these steps to experiment with the visible state type in the alert component:
  - Open **Alert.tsx** and hover over the visible state variable to determine its inferred type.
  - Remove the initial value of true passed into useState. Then, hover over the visible state variable again.
  - The type of the state can be explicitly defined using a generic argument on useState.

```
const [visible, setVisible] = useState<boolean>();
```

  - Restore the **useState** statement to what it originally was, with it initialized as true and no explicit type.

```
const [visible, setVisible] = useState(true);
```

  - Stop the app from running by pressing Ctrl + C.

Creating a React and TypeScript component

Using React DevTools

- React DevTools is **a browser extension available for Chrome and Firefox**. The tools allow **React apps to be inspected and debugged**.
- Carry out the following steps to explore the tools:
  - Start the app in development mode by running **npm run dev** in a terminal
  - Open the browser's development tools by pressing F12. React DevTools adds two panels called **Components** and **Profiler**.

Creating a React and TypeScript component

Using React DevTools

- Carry out the following steps to explore the tools:
  - Select the **Components** panel > select the **Alert** component in the component tree.
  - Notice that the state isn't named – it has a generic name, State. Click the **wand icon** at the right of the hooks section.
  - In Visual Studio Code, open **App.tsx** and pass the **closable** prop to Alert. The app refreshes and the close button appears.
  - Click the close button and notice that the visible state changes to false in DevTools.

Creating a React and TypeScript component

Using React DevTools

- Carry out the following steps to explore the tools:
  - Refresh the browser so that the alert appears again. While still in the **Components** panel in React DevTools, open the settings by clicking the **cog** icon. Tick the "**Highlight updates when components render."** option in the **General** section, if not already ticked.
  - Before we try the re-render highlight, open **Alert.tsx** and update it to render **Gone**! When the **visible state is false** (check visible and return null).
  - Click the close button on the alert in the browser. The re-rendered alert component will be highlighted with a green border.

Creating a React and TypeScript component

Using React DevTools

- **Profiler** tool is helpful in quickly spotting the slow components for a particular user interaction.
- Carry out the following steps to explore the tools:
  - Select the **Profiler** panel. This tool allows interactions to be profiled, which is useful for tracking performance problems.
  - Click the **Start profiling** option, which is the **blue circle** icon.
  - Click the **close button** in the alert.
  - Click the **Stop** profiling option, which is the **red circle** icon. A timeline appears of all the component re-renders.
  - Undo the change we made to the Alert component so that it renders **null** again when invisible.

Creating a React and TypeScript component

Recap

- A type can be added to component props to make them type-safe
- A state can be inferred from its initial value but can be explicitly defined using a generic argument on **useState**
- React DevTools can be installed in a browser to inspect the component tree in a running app and help track down performance problems


Q&A