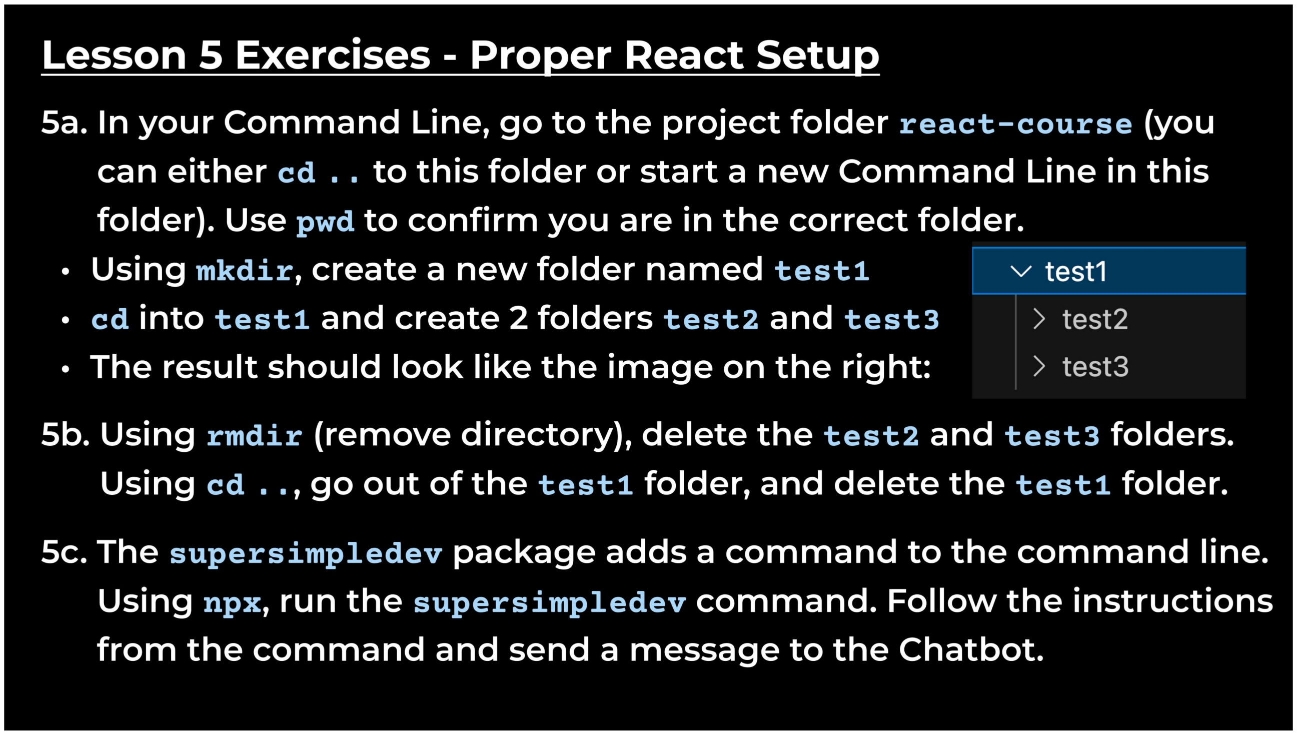


Lesson 5 Exercises - Proper React Setup

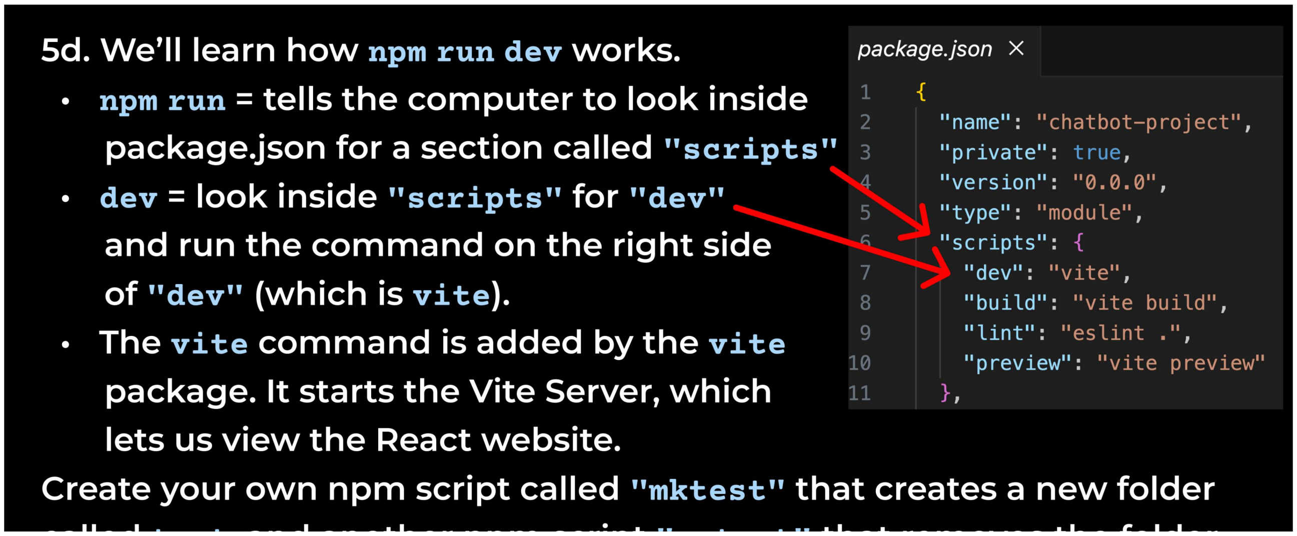
5a. In your Command Line, go to the project folder `react-course` (you can either `cd ..` to this folder or start a new Command Line in this folder). Use `pwd` to confirm you are in the correct folder.

- Using `mkdir`, create a new folder named `test1`
- `cd` into `test1` and create 2 folders `test2` and `test3`
- The result should look like the image on the right:

test1
test2
test3

5b. Using `rmdir` (remove directory), delete the `test2` and `test3` folders. Using `cd ..`, go out of the `test1` folder, and delete the `test1` folder.

5c. The `supersimpledev` package adds a command to the command line. Using `npx`, run the `supersimpledev` command. Follow the instructions from the command and send a message to the Chatbot.



5d. We'll learn how `npm run dev` works.

- `npm run` = tells the computer to look inside package.json for a section called `"scripts"`
- `dev` = look inside `"scripts"` for `"dev"` and run the command on the right side of `"dev"` (which is `vite`).
- The `vite` command is added by the `vite` package. It starts the Vite Server, which lets us view the React website.

Create your own npm script called `"mktest"` that creates a new folder

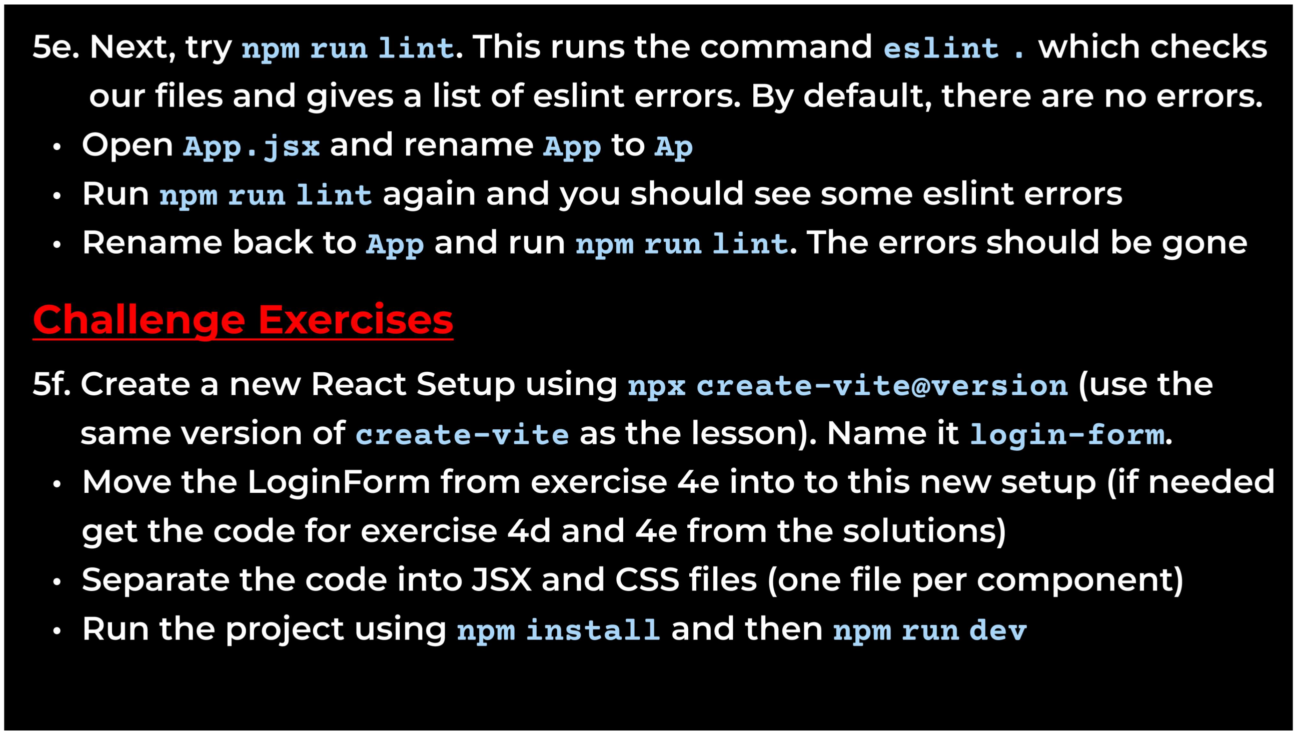
package.json

```
{
  "name": "chatbot-project",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
```

react-course / 1-exercise-solutions / lesson-05 /

↑ Top

Solutions in description



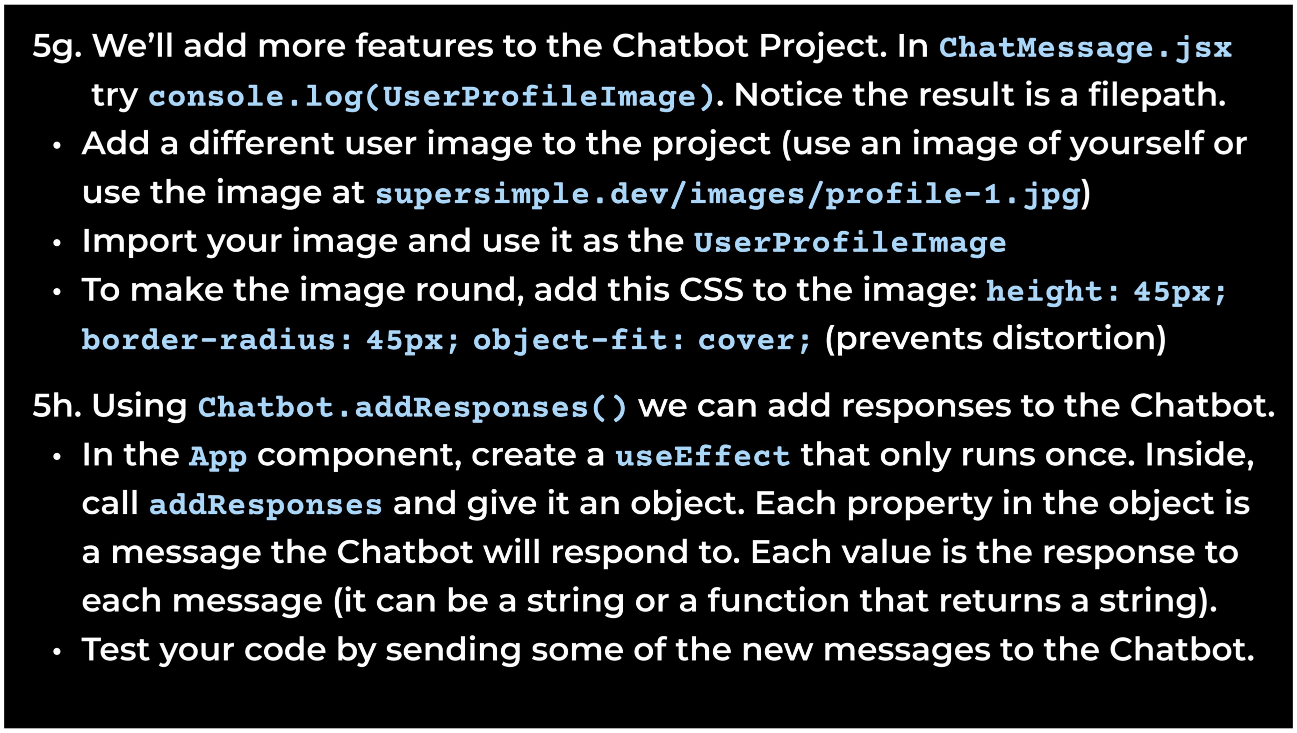
5e. Next, try `npm run lint`. This runs the command `eslint .` which checks our files and gives a list of eslint errors. By default, there are no errors.

- Open `App.jsx` and rename `App` to `Ap`
- Run `npm run lint` again and you should see some eslint errors
- Rename back to `App` and run `npm run lint`. The errors should be gone

Challenge Exercises

5f. Create a new React Setup using `npx create-vite@version` (use the same version of `create-vite` as the lesson). Name it `login-form`.

- Move the LoginForm from exercise 4e into to this new setup (if needed get the code for exercise 4d and 4e from the solutions)
- Separate the code into JSX and CSS files (one file per component)
- Run the project using `npm install` and then `npm run dev`



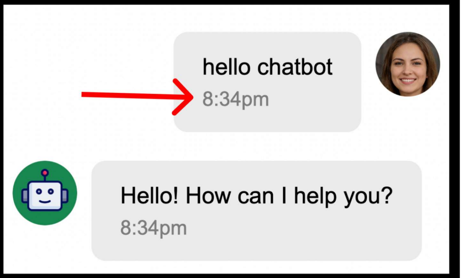
5g. We'll add more features to the Chatbot Project. In `ChatMessage.jsx` try `console.log(UserProfileImage)`. Notice the result is a filepath.

- Add a different user image to the project (use an image of yourself or use the image at `supersimple.dev/images/profile-1.jpg`)
- Import your image and use it as the `UserProfileImage`
- To make the image round, add this CSS to the image: `height: 45px; border-radius: 45px; object-fit: cover;` (prevents distortion)

5h. Using `Chatbot.addResponses()` we can add responses to the Chatbot.

- In the `App` component, create a `useEffect` that only runs once. Inside, call `addResponses` and give it an object. Each property in the object is a message the Chatbot will respond to. Each value is the response to each message (it can be a string or a function that returns a string).
- Test your code by sending some of the new messages to the Chatbot.

5i. Using npm, install the `dayjs` package. Using `dayjs`, add a time under each message that shows when the message was sent. Hints:


hello chatbot
8:34pm
Hello! How can I help you?
8:34pm

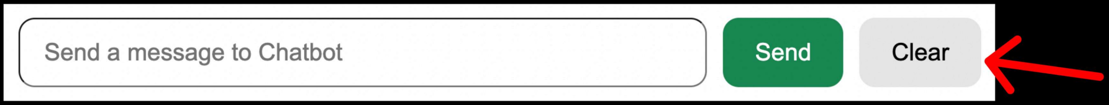
- To import: `import dayjs from 'dayjs';`
- To get the current time (in milliseconds): `const time = dayjs().valueOf();`
- To display the time as a string: `dayjs(time).format('h:mma')`. Or if you use 24-hour format: `dayjs(time).format('HH:mm')`
- Don't forget to save the time whenever you create a new message.
- Also, if you have any default messages at the start of the App, make sure to add a time to each of the default messages.
- Try out your code. Send some messages and check if they have times.

Solutions in description

5j. Save the messages in localStorage so they don't reset when refreshing.

- In `App`, add a `useEffect` that runs whenever `chatMessages` changes.
- Inside the `useEffect`, save the chat messages to localStorage using: `localStorage.setItem('messages', JSON.stringify(chatMessages))`
- In `useState`, load the messages from localStorage and use them as the starting state: `JSON.parse(localStorage.getItem('messages'))` (Initially, there are no messages saved in localStorage, so you'll also need to use `||` to set a default value if no messages exist).

5k. Create a "Clear" button. Clicking it removes all chat messages on the website and updates `'messages'` in localStorage to `[]`.


Send a message to Chatbot
Send
Clear