Javaid Karim – CTO of Nester (who came to talk in UX/UI week)

He has grown up through the industry so has seen changes come and go.

Why was React invented?

Brief history of world wide web:

- WWW first invented – just used HTML
- Then came styling to make it look better – CSS. Original CSS had problems which is why variations were introduced: SASS, SCCS, BOOTSTRAP
- Interactivity needed – JAVASCRIPT
- Browser wars:
    - INTRANETS
    - ASP/PHP
    - BROWSERS – NETSCAPE, IE, FIREFOX, CHROME, SAFARI. Each with their own problems
- Above was trying to solve problems on the server, so more front end
    - JQUERY
- So more front end technologies took off (allowing for single-page applications):
    - ANGULARJS - Google
    - VUE.JS – Evan You – tried to take the best of Angular. Very popular in Asia
    - KNOCKOUT.JS - Microsoft
    - ANGULAR 2+ - Google
    - REACT - Facebook

All new technologies try to solve a problem but also cause more problems!

Javaid hated React at first! Had used all the others from the Front-End list, really liked Vue. Then came back to React in 2018 and really got into it.

Javaid actually prefers to have his CSS in same file as component, rather than separate files. He finds it easier to debug

Principles when building a React app

"Systems are not complicated, people are!" Break it down to small problems and tackle each step.
From his slide:

- KISS, YANGNI and SOLID/SRP
    - Keep It Simple, Stupid! – self-explanatory!
    - You are not gonna need it – don't second guess what it's needed for
    - Solid = Javaid only talked about the first S (which is SRP) but I've copied and pasted from Wikipedia:

        - The **S**ingle-responsibility principle: "There should never be more than one reason for a class to change." In other words, each part should only do one thing
        - The **O**pen–closed principle: "Software entities ... should be open for extension, but closed for modification."
        - The **L**iskov substitution principle: "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it."
        - The **I**nterface segregation principle: "Many client-specific interfaces are better than one general-purpose interface."
        - The **D**ependency inversion principle: "Depend upon abstractions, [not] concretions."
- React is a SPA (Single-page application)
- React is a view library – it's just for building components – you need other libraries for things like animation etc
- Other libraries you may wish to use for different jobs:
    - State Management – useState, Context, Redux, Mobx
    - Forms – custom, Formik, React hook form
    - Validation – custom, JOI, Yup
    - UI Frameworks – Material UI, Bootstrap, Semantic UI, Chakra UI
    - Styling – CSS in JS, Tailwind
    - APIs – fetch, axios
    - Server Side Rendering Next.JS – performance, SEO etc

Always consider library size and use the right tool for the job! Remember to look at how often libraries are maintained! What are the docs like? How quickly do they respond to issues?

Javaid recommends going through the pain of building your own form! It will make you appreciate the libraries of forms. Also learn from their GitHub about how they're set up.

Forms and validation often go hand-in-hand – Javaid likes Formik combined with Yup.

UI frameworks give you a lot of components out of the box to save you a lot of time coding – Material UI is very popular for this.

Single page applications solved one problem (e.g. scalability) but then brought in new problems – if you use too many libraries then your page takes ages to download on slower connections (but will be fast once downloaded)

Questions at this point:

(Selection that felt relevant)

How do you choose libraries? = Think about problem you are trying to solve! Looks at library sizes and also consider how much of the library you are using. Could you make your own component that does the same or a similar thing?

Is it ok to copy others work? = Make sure you attribute where necessary, but open source is about sharing solutions

How do you identify a bad library? = Import it and stress test it in isolation before building your whole app around it! Keep an eye on how often it is updated etc. If you have to migrate after discovering one – do it step-by-step in small increments and test each bit to ensure nothing breaks!

In the industry, how do you allow people to learn new technologies when they come out? = Javaid sets himself an objective to have a look at new technologies and learn the basics (personally). But you have to sell it business management – what are the benefits? Get trainers in or send developers on training courses.

Other key points:

Businesses only know what they want after you've built the first version of it!!
Use your big 4 week project in your interview – learning experiences, "If I had more time", once again focussing on the process rather than the finished product!