Part 1 - Basic React

1) Start with first 3 slides from presentation explaining the basics points of React. (features, advantages, use cases, es6, declarative) https://docs.google.com/presentation/d/1Nffvb2Vs6ORTSjbwQJxNbXU72x6Z3ghuJzHVFKbrgSs/edit?usp=sharing
2) Look at the basics of ES6 at https://medium.com/the-andela-way/a-beginners-guide-to-react-with-es6-a2ed0b5c977e
3) Create new project with create-react-app https://reactjs.org/docs/create-a-new-react-app.html
4) Analyze the folder structure. Run "yarn build" and see what's produced. Talk about why it's an overkill to use react just for small widgets (mention Preact as alternative for that).
5) Explain what we'll be doing (https://blog.ferriesingreece.com/wp-json/wp/v2/posts/)
6) Edit App.js by adding a card from https://materializecss.com/cards.html (leave the HTML intact to see warnings). Explain JSX (also use babel transpiling).
7) Add css from https://materializecss.com/getting-started.html and see result on page
8) Add a 2nd copy of the HTML of a post - or more.
9) Create a Post component with the post HTML (without adding export default)
10) Add Post component to App.js (without adding import) and resolve issues. (also comment on syntax with first capital letter and enclosing tags).
11) Explain props and add props for image, title, excerpt, link (create link as *<a>* for starters) from
12) Go to Post.js and pass the props to the render function (also add .card .card-content p {height: 112px;} & .card .card-image .card-title{right: 0; background-color: rgba(0,0,0,0.3);} to css).
13) Replace react default HTML with materialize navbar with badge from "Badges in Navbar" https://materializecss.com/badges.html
14) Conditional rendering - create a renderNotifications() function and let newNotifications = 4; so by checking with an if statement, it renders the *<span>* with the badge or not (try without returning the value in the function first).
15) Create a clearNotifications() function and call it from an onClick event (try {this.clearNotifications()} first to show the difference between calling it and referencing it with {this.clearNotification}). Change the value of the let "newNotifications" to 0 and show that nothing happens. Explain what state is.
16) Create a constructor and explain briefly what it is, set the initial state and try to change now the value of the state directly (mutation). Then call setState. See error and then bind "this". https://medium.freecodecamp.org/this-is-why-we-need-to-bind-event-handlers-in-class-components-in-react-f7ea1a6f93eb

Part 2 - Making our App dynamic

17) Remind passing props from parent component downwards. Open community project and show example of Post.js `toggleComment()` function called from PostInteractions.js to pass data from child component upwards.
18) Explain that sometimes we need to pass data from child components upwards, or even far away into the component tree. Introduce redux and its usefulness, especially with ajax requests. Redux flow -> https://github.com/buckyroberts/React-Redux-Boilerplate (Slides 4-6 of presentation)
19) Install redux (`npm install --save react-redux`) and create folders "actions" and "reducers" with index.js in them, also move all components to "components" folder if we haven't done so. Make necessary changes to root index.js file to make use of redux in the application (wrap *<Provider store={store}>* on the application root component, import Provider from react-redux, createStore from redux, and rootReducer from the reducers folder. Use `const store = createStore(rootReducer)` to create store without middleware for starters.
20) Explain how why will do our http requests using axios instead of Fetch API (slide 8,9). Install axios (`npm install axios`), then go to actions/index.js and import it. Type your function (e.g. `getPosts()`), make the request with `axios.get()` and pass it to a const. Use `.then()` and dispatch the action type/payload.
21) Create a new reducer (e.g. "reducer_posts") and import the function GET_POSTS. Explain why a switch is useful in a reducer, that an action is being "heard" from all reducers etc. Go to index.js of reducers folder and import the new reducer we created, passing the payload as posts.
22) Go to App.js and import the action, trying to call it at once without connecting it with mapDispatchToProps to get error.
23) Install redux thunk and apply middleware to the store.
24) Change post reducer to store only the data needed for rendering the posts.
25) Remove static Components rendering in App.js and replace them with a map function returning a *<Post>* component for each post in the application state with dynamic content. Watch how Wordpress API returns html for the excerpt and not plain text. Use `dangerouslySetInnerHTML` (https://reactjs.org/docs/dom-elements.html#dangerouslysetinnerhtml) to resolve the issue and mention the danger of XSS attacks.

Part 3 - Create Routes

26) Create a new component and name it something like “SinglePost” or “PostPage” or something. Put some static HTML from other components or from the materialize framework examples.
27) Install `react-router-dom` (and not react-router)
    https://reacttraining.com/react-router/web/guides/quick-start
28) Show slides concerning the router. Don’t forget to mention route priorities with switch.
29) Go to root index.js file and add a `<BrowserRouter>`, a `<Switch>` and 2 routes. One for the main page (“/”) and one for the posts (“`/posts/:slug`”). Import the necessary Singlecomponents.
30) Try hitting a url that should exist on the browser. The component should attempt to mount, but some functions that were needed in the HTML we tried to render are missing.
31) Create a new Header component including the HTML and the related functions and state needed. Then remove them from the route components.
32) Add Header component to root index.js, outside of *`<Switch>`* so that it becomes common for all routes and doesn’t need rerendering. Now switch and header components need to be wrapped inside a *`<React.Fragment>`* to avoid extra html nodes.
    https://reactjs.org/docs/fragments.html
33) Pass {post.slug} data to Post.js component. `import { Link } from 'react-router-dom'` and replace *`<a>`* element with react-router *`<Link to={} >`*.
34) Discuss about the redux state and what would be the best way to render single posts. Some common cases:
    a) Also store the full text in redux state on initial `getPosts` request, then use that state to render the page. In this case we’d later need to check if the post is not initially loaded (e.g. user went to that url directly).
    b) Use the post ID to make another request to the API and get the data of the post again, this time store it in a reducer with a name like “`selectedPost`” or something.

    The first case has some code difficulties, like creating a more complex data structure (need to iterate through objects), while the second case makes the application a bit slower, often requesting the same data we already have in store.

    Resolving with case (b):

35) Add a console.log in `componentDidMount` in SinglePost.js and view the “`this.props`” that comes from the react router. We can use this info (slug) to get the post data and render it. Create a new action `getCurrentPost`() to do that. Add action creator in actions index.js, connect redux with SinglePost.js, create a new reducer “reducer_current_post.js” and combine the new reducer state with the app state.

    Request by slug -> `axios.get(`https://blog.ferriesingreece.com/wp-json/wp/v2/posts/?slug=${slug}`);`

36) Create a chained request, after getting the response from request by slug, that uses the ID of the post and gets the featured media from the wordpress API.
37) Update the SinglePost.js props to show the content.
38) Dispatch an action that sets a redux state while loading the post. Use that state to conditionally render a preloader (https://materializecss.com/preloader.html) instead of the post html when the post we want to view is different than the last viewed.