# Theoretical Part

**- What is React JS?**
React es una librería basada en componentes con estados que permiten controlar el renderizado de partes específicas del DOM.

**- Why should we use ReactJS?**
Precisamente, al estar basado en componentes, permite crear aplicaciones fácilmente escalables, reutilizando estos mismos. También es un framework actual con las ventajas que eso conlleva, como un buen mantenimiento de las principales librerías así como frecuentes actualizaciones o mejoras como por ejemplo los Hooks. También al usar React se están dando pasos en dirección del desarrollo de apps pues React Native está basando en React y comparte gran parte de su lógica.

**- What are the life stages of a component?**
Son los estados necesarios para que un componente pueda montarse en el DOM y renderizarse si tiene un cambio de estado. Los podemos desglosar en montaje, actualización y desmontaje.

**- How does ReactJS use "keys"?**
Las keys son identificadores de elementos necesarios para que no quede ningún elemento sin identificar. Es típico tenerlo en cuenta a la hora de usar un map que genere alguna etiqueta. Si no añadimos keys React no puede distinguir entre los elementos creados.

**- Is there a difference between "container component" and "presentation component"?**
Una manera metafórica de explicarlo es que un container component se encarga de ir a buscar las pinturas, el lienzo, el lugar donde pintar, la hora, el orden y las condiciones y el presentation component es un pintor que pertenece a un género y si por ejemplo, es cubista, dibujara cuadros cubistas, pero que variarán según lo proporcionado por el container component.

**- Why are "synthetic events" used?**
- What is an "arrow function"?
Es una manera muy sencilla de declarar funciones en forma de expresión, de tal manera que cuando son llamadas, recogen en ámbito desde donde son llamadas sin necesidad de usar “binds” de tal manera que simplifica mucho el código.

**- What is React Redux?**
Redux es un contenedor de estados globales. Del mismo modo que los componentes tienen estados, estos ‘de entrada’ solo pueden propagar datos o funciones de padres a hijos o de hijos a padres. A menudo es necesario poder acceder a un dato que no se encuentra colindante al componente que lo solicita, por ello la store de Redux almacena globalmente estos estados para que puedan ser conocidos por el resto de componentes que se conecten a la dicha store.