***Glosario de conceptos utilizados en clase***

**React JS:** es una biblioteca para desarrollo web, por lo cual debemos contar con conocimientos mínimos sobre los lenguajes que el navegador web interpreta.

**HTML:** es un lenguaje de etiquetas, el cual dará la estructura para nuestras páginas web.

**HTML 5:** es una nueva versión de diversas especificaciones, entre las que se encuentran: HTML 4-XHTML 1-CSS Nivel 2-DOM Nivel 2 (DOM = Document Object Model).

**DOCTYPE:** no es una etiqueta, sino una instrucción para indicar al navegador qué versión de HTML vamos a utilizar.

**OM (Document Object Model o modelo de objetos del navegador):** nos sirve para acceder a cualquiera de los componentes que hay dentro de una página.

**CSS (cascading style sheets - hojas de estilo en cascada):** es un lenguaje de diseño gráfico con el cual podremos dar estilos (diseño, colores, márgenes) a nuestras webs desarrolladas con HTML.

**Javascript:** es el lenguaje de programación web por excelencia. Decimos que se trata de un lenguaje de programación interpretado.

**Virtual DOM:** Es un patrón de comportamiento y React lo implementa con una tecnología llamada “Fiber”. En sí, resulta ser todo lo que React sabe de tu aplicación y cada nodo o fibra.

**Node.js:** es un entorno de ejecución de javascript que le permite al código en js ser ejecutado en nuestra computadora.

**NPM (Node Package Manager):** cuando usamos Node.js, rápidamente tenemos que instalar módulos nuevos (librerías), ya que al ser un sistema fuertemente modular viene prácticamente “vacío”. Por lo tanto, para utilizar una funcionalidad de alguna librería publicada, deberemos instalar módulos adicionales.

Esta operación se realiza de forma muy sencilla con esta herramienta.

**CLI:** la interfaz de línea de comandos o interfaz de línea de órdenes es un método que permite a los usuarios dar instrucciones a algún programa informático por medio de una línea de texto simple.

**Sugar Syntax:** refiere a la sintaxis agregada a un lenguaje de programación con el objetivo de hacer más fácil y eficiente su utilización. Favorece su escritura, lectura y comprensión.

**Webpack:** es un module bundler o empaquetador de módulos.

**Eject:** es una acción permanente, que permite tener un control más específico del bundling, a costa de que de ahora en adelante tendremos que encargarnos de mantenerlo.

**Transpiling:** es el proceso de convertir código escrito en un lenguaje, a su representación en otro lenguaje.

**JSX:** Es una extensión de sintaxis de Javascript que se parece a HTML. Oficialmente, es una extensión que permite hacer llamadas a funciones y a construcción de objetos. No es ni una cadena de caracteres, ni HTML

**Componentes:** básicamente, las aplicaciones en React básicamente se construyen mediante los mismos. permiten separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada.

**Propiedades:** son la forma que tiene React para pasar parámetros de un componente superior a sus children.

**Componentes de presentación:** son aquellos que simplemente se limitan a mostrar datos y tienen poca o nula lógica asociada a manipulación del estado (por eso son también llamados stateless components).

**Componentes contenedores:** tienen como propósito encapsular a otros componentes y proporcionarles las propiedades que necesitan. Además se encargan de modificar el estado de la aplicación para que el usuario vea el cambio en los datos (por eso son también llamados state components).

**Children:** es una manera que tiene react de permitirnos proyectar/transcluir uno o más componentes dentro otro.

**Ciclo de vida:** no es más que una serie de estados por los cuales pasa todo componente a lo largo de su existencia. Esos estados tienen correspondencia en diversos métodos, que podemos implementar para realizar acciones cuando se van produciendo.

**Métodos de ciclos de vida (class based):**

* **componentWillMount()\*:** este método del ciclo de vida es de tipo montaje. Se ejecuta justo antes del primer renderizado del componente.
* **componentDidMount():** método de montaje, que solo se ejecuta en el lado del cliente. Se produce inmediatamente después del primer renderizado. Una vez se invoca este método ya están disponibles los elementos asociados al componente en el DOM.
* **componentWillReceiveProps():** método de actualización que se invoca cuando las propiedades se van a actualizar, aunque no en el primer renderizado del componente, por lo tanto no se invocará antes de inicializar las propiedades por primera vez.
* **shouldComponentUpdate (nextProps, nextState):** es un método de actualización y tiene una particularidad especial con respecto a otros métodos del ciclo de vida, que consiste en que debe devolver un valor booleano.
* **componentWillUpdate (nextProps, nextState):** este método de actualización se invocará justo antes de que el componente vaya a actualizar su vista.
* **componentDidUpdate (prevProps, prevState):** método de actualización que se ejecuta justamente después de haberse producido la actualización del componente.
* **componentWillUnmount():** este es el único método de desmontado y se ejecuta en el momento que el componente se va a retirar del DOM.

**Promise:** es un objeto que permite representar y seguir el ciclo de vida de una tarea/operación (función).

**Map:** es un método que nos permite generar un nuevo array, tomando de base otro, y utilizando una función transformadora.

**Modelo cliente-servidor:** establece que los distintos consumidores se identifican entre ellos y acuerdan una manera de transferir la información.

**Push:** nace para poder generar engagement, y lograr que los usuarios recuerden que nuestra app existe, y que puede proveerles con algo que les pueda interesar, en el momento en el que el servidor considere oportuno.

**Requests via http/s:** vienen para ayudarnos a realizar una solicitud a un servidor, y nos permiten establecer un protocolo de transferencia.

**URL y VERB:** nos permiten definir una manera de explicarle al servidor la dirección y nuestras intenciones.

**Query params:** nos permiten incluir en la dirección información que se usa para especificarle al receptor parámetros para efectuar una búsqueda.

**URL params/segment:** son una convención para incluir el identificador del recurso dentro de la misma url, son más comunes cuando ya se conoce el recurso específico que se buscará.

**Body:** se utiliza para transferir piezas de información entre el cliente y el servidor.

**Headers:** se usan para definir las respuestas soportadas, requeridas o preferidas. También para agregar información extra y lo que desees en forma de texto.

**Fetch:** podemos hacer un request de manera simple, utilizando Fetch API. Esta nos provee con una promesa, que se resuelve al terminar el request.

**NavLink:** es un link con un estilo, está siempre detectando la ruta actual, y si coincide con la suya nos activa la clase que le demos para que el user sepa qué ítem de la lista corresponde con la vista actual.

**useParams:** lo podemos utilizar para leer en js los parámetros de la ruta. En combinación con un useEffect, nos sirve para obtener actualizaciones sobre los parámetros.

**Evento:** es un estímulo programático, que puede ser provocado de manera automática, o ser el resultado de una interacción del usuario con la UI.

**Event Listener:** es un patrón de diseño que sirve, como su nombre lo indica, para escuchar cuando un algo ocurre en algún elemento, librería o API, y poder realizar una acción en consecuencia.

**Firebase:** es un servicio provisto por Google para satisfacer las distintas necesidades que puede tener una aplicación y su ciclo de desarrollo

**Colecciones (Firestore):** son contenedores de documentos, los cuales se agrupan obligatoriamente dentro de ellas. Dichos documentos pueden, a su vez, tener colecciones dentro.