En el siguiente capítulo se detallan aspectos correspondientes al diseño e implementación del trabajo. En la sección 5.1 se presentan las tecnologías utilizadas para la generación de los datos de las distintas variaciones que un usuario desea configurar. En la sección 5.2 se presentan las tecnologías utilizadas para la generación de los escenarios a partir de los ingresos del usuario.

**5.1 Implementación FrontEnd**

**5.1.1 Tecnologías**

Para comenzar con la técnica de Scheduling, previamente se tuvo que crear una interfaz gráfica Web, donde el usuario va a indicar que parámetros quiere variar sobre un escenario que fue generado en las pantallas de Simugan. Estas pantallas tienen como objetivo principal, recopilar la información del interés del usuario dándole de forma rápida y sencilla una forma de poder ver los resultados de las distintas variantes del escenario principal.

Dicha información viaja a un servidor, que primero se encarga de generar todos los posibles escenarios en base a los datos que el usuario desea variar, y luego esos escenarios son valorizados por un algoritmo de Scheduling, donde el más prioritario será el que tenga un valor mayor, el cual será seleccionado y enviado al Simulador Simugan.

Para desarrollar las pantallas, tuvimos que decidir si usar ReactJS o Angular que son tecnologías de punta en el mercado.

**ReactJS** es una librería Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Fue creada para manejar componentes con estados para agilizar el desarrollo partiendo del concepto de reutilización de componentes.

**Angular** es un framework de desarrollo para JavaScript creado por Google. La finalidad de Angular es facilitarnos el desarrollo de aplicaciones web y además darnos herramientas para trabajar con los elementos de una web de una manera más sencilla y óptima.

Acá es donde vemos **la primera gran diferencia** entre ambas tecnologías. React es una librería y Angular es un framework.

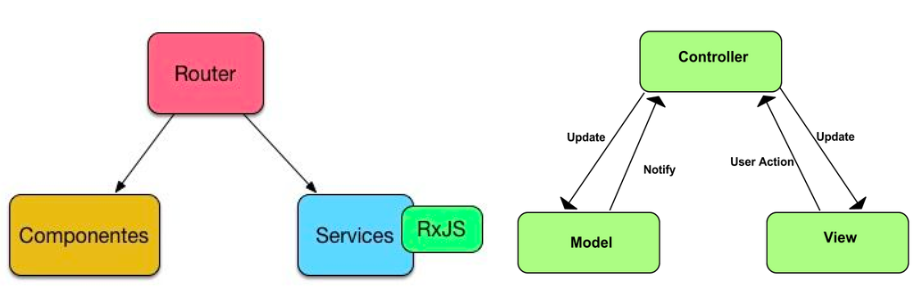
Por sí solo, React no es capaz de almacenar los estados consistentes dentro de una aplicación, por lo que necesita otra herramienta para no perder información. Angular por su parte, tiene mecanismos para almacenar dichos estados en memoria.

Hubo que buscar alternativas para administrar los estados en React ~~antes de elegir una tecnología~~, por lo que surgió Redux.

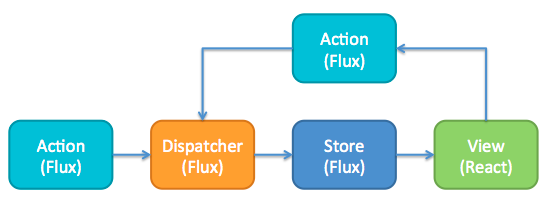
**Redux** se define como una herramienta (librería) para la gestión de estados, que fue creada en 2015 por Dan Abramov. Define “estado” al conjunto de todos los valores almacenados por una aplicación mediante propiedades o variables.

Como **segundo punto** de evaluación, ReactJS + Redux tienen un peso de de 150KB, contra los casi 600KB que pesa Angular.

Como **tercer punto** de evaluación se tomaron las arquitecturas que utilizan. Angular maneja una arquitectura básica MVC (Model View Controller), donde disponemos de componentes para la vista, enrutador para la capa de control y servicios para la capa de backend. Se utiliza un paradigma orientado a componentes.



En cambio React maneja una arquitectura llamada Flux, que es similar a MVC ya que también contiene su modelo, vista y controladores pero esta pensada en un flujo de datos unidireccional. Los datos viajan desde la vista por medio de acciones para mantener el estado de la aplicación. Aquí es donde se combina con Redux incorporando un Store que se va a modificar mediante acciones.



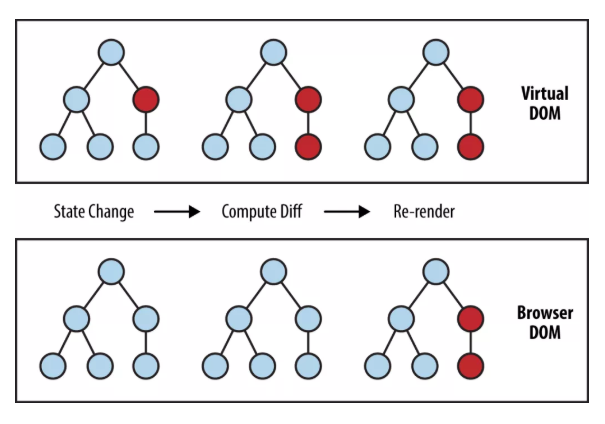
Como **quinto punto** de evaluación Angular centra sus plantillas en HTML, donde escribimos la lógica en HTML trasladando javascript a HTML. Esto implica que se mantiene un HTML y javascript por componentes lo cual da más claridad de las cosas pero la detección de errores en una plantilla se produce en tiempo de ejecución, aportando además una información poco determinante para encontrar el error. En cambio, React maneja toda la lógica y vista en javascript, por lo que se traslada HTML a javascript. Esto genera código más centralizado pero pueden ser archivos bastante grandes por componentes, pero la detección de errores se genera en la compilación de la plantilla, aportando información acerca del error y la línea que provoca el error.

*Por todas estas evaluaciones, se decidio elegir* ***ReactJS + Redux*** *para el desarrollo de la aplicación.*

**React** fue creada por Jordan Walke, un ingeniero de software en Facebook, inspirado por los problemas que tenía la compañía con el mantenimiento del código de los anuncios dentro de su plataforma. Por lo tanto, podemos encontrarla en páginas desarrolladas íntegramente en **ReactJS** como Facebook e Instagram. Es mantenido por Facebook y la comunidad de software libre.

React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. Cada pieza es un componente y cada componente tiene un estado. Está construido en torno a hacer funciones, que toman las actualizaciones de estado de la página y que se traduzcan en una representación virtual de la página resultante. Siempre que React es informado de un cambio de estado, vuelve a ejecutar esas funciones para determinar una nueva representación virtual de la página, a continuación, se traduce automáticamente ese resultado en los cambios del DOM necesarios para reflejar la nueva presentación de la página. Utiliza un concepto llamado DOM virtual que hace selectivamente sub-árboles de los nodos sobre la base de cambios de estado, desarrollando esto, con la menor cantidad de manipulación DOM posible, con el fin de mantener los componentes actualizados.

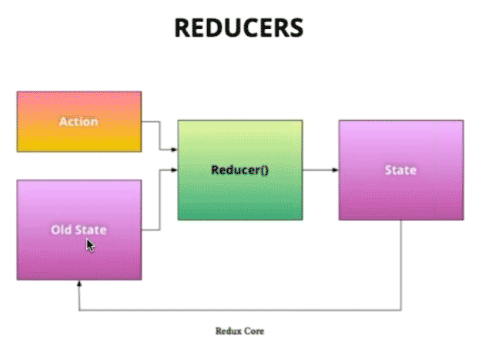
Para cambiar los elementos del DOM, **ReactJS** ejecuta el algoritmo **diffing**, que identifica los cambios sobre el objeto a renderizar. Luego se actualiza el DOM con los resultados de la ejecución del algoritmo de diffing. Para entender mejor como funciona el DOM Virtual de React, damos un ejemplo. Supongamos que tenemos un auto donde tenemos todas las propiedades que este puede tener y lo renderizamos en el DOM. Luego, se nos ocurre cambiarle el color, por lo que en este caso React interpreta que hay cambios, ejecuta el algoritmo de diffing y le dice al DOM que el auto ha cambiado de color, por lo que solo tiene que renderizar este cambio.

Con el nuevo árbol calculado, se vuelve a ejecutar el método render, el cual va a volver a pintar en pantalla los cambios generados por el diffing

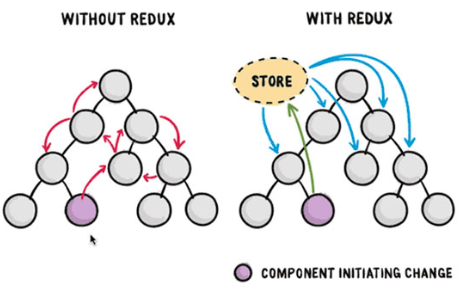
En cuando a **Redux** podemos decir que es un patrón de arquitectura de datos que permite manejar el estado de la aplicación de manera predecible. Está pensado para reducir el número de relaciones entre componentes de la aplicación y mantener un flujo de datos sencillo. Fue iniciado por la comunidad de React como evolución y mejora a las ideas de Flux, quien es capaz de adaptarse a cualquier tipo de librería o framework del lado del cliente. Se caracteriza por ser una librería realmente liviana, porque pesa apenas 2Kb. Esto, lo llevo a ser adaptado con éxito a grandes empresas como por ejemplo Netflix y Facebook. Esta última lanzo Flux que es el pilar desde donde parte Redux que implementa el patro de Flux. Redux almacena sus “estados” en un Store, que es el estado al que todos atienden. Cada componente que necesite los datos para renderizar, acudirá al Store para recuperarlos. Los datos del Store viajan a los componentes en una sola dirección. Esto nos dice que el data-binding que usamos para enviar al store hacia los componentes será de una sola dirección.

Si algún componente necesita modificar el estado, va a realizarlo mediante acciones que son las que representan la única via de producir una modificación en el estado global de la aplicación.

Una vez emitidas las acciones, se procesan por los reducers. Estos reciben dos parámetros, el estado anterior y la acción. Una vez efectuados los cambios, el reducer devuelve un nuevo estado.



A continuación mostraremos un diagrama de cómo se comporta React con y sin Redux



**5.1.2 Generación de datos**

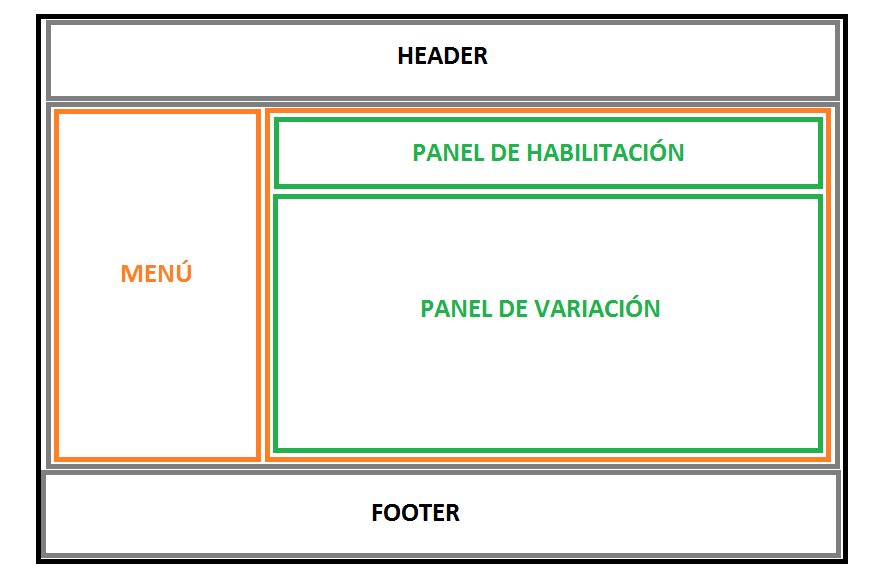
Una vez definidas las tecnologías con las que implementamos la interfaz gráfica de carga, comenzamos con el desarrollo.

Nuestra interfaz cuenta con una distribución de componentes en toda la pantalla. Cuenta con:

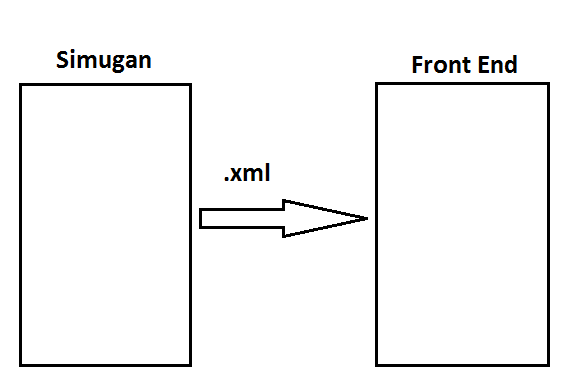
* Un componente Header donde se mostrara el título y los logos en cuestión.
* Un componente Content donde tendrá todo el contenido lógico y el menú de opciones.
* Un componente Footer donde tendrá los datos de la entidad que trabaja y da soporte a la aplicación.

Dentro del componente Content, hacemos una subdivisión de componentes donde vamos a ver del lado izquierdo el menú de opciones que tendrá el usuario habilitadas más la interfaz con la variación de los datos. La variación de los datos también tiene otra subdivisión, donde vamos a tener en la parte superior un componente que maneje la habilitación de la variación de los datos y la cantidad de veces que el usuario va a querer variarlos. En la parte media vamos a encontrar un componente que maneja la lógica de los datos.

En la siguiente imagen mostramos la distribución de los componentes principales de la aplicación.



Los recuadros naranja son los que son alimentados por los datos generados por el usuario mediante la generación de escenarios que tiene Simugan, donde se definen los criterios a tener en cuenta para simular la evolución de un escenario. Dicha información llega a nuestro sistema como un archivo XML con diferentes tags los cuales contienen atributos.



En base a las habilitaciones que tenga el XML se van a mostrar las opciones de variación de parámetros.

Originalmente, los tags que pueden variar en el **MENÚ** son los siguientes:

1. <pasture>: Tasas de crecimiento de recursos forrajeros
2. <stockPilled>: Digestibilidad y rinde de cada diferido
3. <crop\_stubbles>: Digestibilidad y rinde de cada rastrojo
4. <paddocks>Has de cada potrero
5. Mobs
6. <makeSilage> Reglas de ensilaje y masa remanente
7. <sellRule> Peso de venta de invernada, novillos y vaquillonas
8. <feedlot> Pesos de entrada y salida/venta de corrales de crecimiento o engorde
9. <earlyWeaning> Reglas de destete precoz
10. <fattening> Regla de engorde de vacas vacías y/o cuts a pasto o corral

Dentro de cada tag hay varios atributos que van a cambiar en base a la carga del usuario. Si en el usuario en cuestión no cargo originalmente alguno de estos tags, la opción no la vera en pantalla. Por ejemplo, si el usuario cargo la opción 1, 2, 4 y 5, las únicas opciones que podrá modificar en esta nueva interfaz serán Pasture, StockPilled, Paddocks y Mobs. El resto no las tendrá habilitadas.

A sí mismo, el sistema prevé la situación donde el usuario no quiera variar algún parámetro cargado originalmente en Simugan, por lo que se le da la opción de habilitar o no la variación de ese parámetro. En caso de que desee variarlos, se le solicitará la cantidad de veces que quiere hacerlo, por lo que esto estará en la sección **PANEL DE HABILITACIÓN** ~~de la imagen anterior (poner numero de referencia de imagen)~~

Dentro del **PANEL DE VARIACIÓN**, dependiendo de la opción que seleccionará el usuario se mostraran distintos componentes hechos a medida para la carga de los datos. Como valores iniciales, se mantendrán los cargados en el escenario original para ahorrar trabajo en caso de querer variar solo alguno de todos los valores posibles.

Dentro de los componentes de variación tendremos:

* Variaciones mensuales
* Variaciones fijas
* Variaciones de rango mínimo y máximo.
* Tablas con paginados
* ~~AGREGAR LOS OTROS COMPONENTES QUE TENGAMOS~~

**5.1.3 Envío de datos al servidor**

Una vez finalizada la carga de los datos, se prepara un paquete para enviar al servidor (ver 5.2 implementacion Backend) el cual hará de nexo entre Simugan y la generación de escenarios a partir de la simulación original. El envío se hará a través de una conexión http. Este es un protocolo de comunicación que permite transferencias de información. Fue desarrollado por World Wide Web Consortium (con sus siglas W3C). HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web para comunicarse. No tiene estado, es decir, no guarda ninguna información sobre conexiones anteriores. HTTP es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. En este caso, el cliente es la capa Front End de carga de datos y el servidor es la capa extra que se agregó entre el Front End y Simugan. El cliente, realiza una petición enviando un mensaje con cierto formato al servidor. El servidor le envía un mensaje de respuesta con el código correspondiente.

**5.2 Implementación BackEnd**

Para una resolución eficiente, se decidió agregar una capa extra entre las generaciones y el servidor Grid donde corre Simugan. Esta capa se realizó con la tecnología Java.

Se decidió utilizarla, porque da muy buen soporte a la programación concurrente y orientada a objetos, con el fin de que la aplicación sea escalable y mantenible. A su vez, el servidor Grid con el que cuenta Simugan, maneja esta tecnología en su desarrollo, por lo que la comunicación entre ambos sistemas es más legible.

En primera instancia, se decidió implementar una capa extra intermedia, dado que desarrollar la generación de escenarios en el cliente, implica depender de la capacidad de proceso que tenga el usuario en su CPU, por lo que podría variar el rendimiento de un usuario a otro. A se vez, delegando esta responsabilidad al servidor, queda en manos de los administradores poder agregar hardware para procesar datos con mayor complejidad o estudiar otras formas de A su vez, esta capa cuenta con la arquitectura REST.

//Revisar todo esto

La transferencia de estado representacional (REST - Representational State Transfer) fue ganando adeptos como una alternativa más simple al protocolo de acceso a objetos simples (SOAP - Simple Object Access Protocol) y a los servicios web basados en el lenguaje de descripción de servicios web (WSDL - Web Services Descripcion Language) [R. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, Tesis Doctoral 2000.]. La arquitectura REST define un conjunto de principios por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo: cómo acceder al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes [L. Deseta. (2008, Noviembre) Introducción a los servicios web RESTful. Dos Ideas (En Ideas Ágiles)].

Un servicio web se define como una aplicación software identificada por un URI cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML [World Wide Web Consortium (W3C). (2004, Febrero) Web Services. [Online]. https://www.w3.org/TR/ws-arch/]. Los servicios web permiten la interoperación de sistemas distribuidos heterogéneos con independencia de las plataformas hardware y software empleadas. Por lo tanto, puede pensarse en ellos como en una arquitectura, conceptual y tecnológica, haciendo posible que distintos servicios se describan, publiquen, descubran y utilicen a través de sistemas distribuidos, empleando la infraestructura proporcionada por Internet [J. J. Domínguez Jiménez, A. Estero Botaro, I. Medina Bulo, M. Palomo Duarte, and F. Palomo Lozano, "El reto de los servicios Web para el software libre," in Proceedings of the FLOSS International Conference, 2007, pp. 117 - 132.]. REST emergió en siglo XXI como el modelo predominante para el diseño de servicios. Ha logrado un gran impacto en la web desplazando a SOAP y a las interfaces basadas en WSDL por tener un estilo más simple de usar [L. Deseta. (2008, Noviembre) Introducción a los servicios web RESTful. Dos Ideas (En Ideas Ágiles).]. Fielding define a REST como “un estilo de arquitectura de software para sistemas hipermedia distribuidos tales como la World Wide Web” [R. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, Tesis Doctoral 2000.]. Para otros autores, es un conjunto de principios para el diseño de redes, que es utilizado comúnmente para definir una interfaz de transmisión sobre HTTP de manera análoga a como lo hace SOAP [C. Morales Machuca, "Estado del arte: Servicios Web," Universidad Nacional de Colombia, Tesis de Maestría 2010]. Una implementación de un servicio REST cuenta con los siguientes principios [L. Richardson and S. Ruby, RESTful Web Services. California, USA: O'Really Media, Inc., 2008.]:

* **Verbos HTTP**: los más comunes son GET, POST, PUT y DELETE [R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T. Fielding, Hypertext transfer protocol--HTTP/1.1, 1999.].
* **Sin estado:** ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.
* **Identificadores de recursos uniforme (URI):** cada recurso es accedido únicamente a través de su URI.
* **Hipermedios:** permite transferir los formatos HTML, JSON y XML.

//Hasta aca, salio de: <http://sedici.unlp.edu.ar/bitstream/handle/10915/56635/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y>

Esta capa recibe la información en formato JSON. En el paquete estará la información proveniente de la entrada de Simugan, los datos del usuario y las variaciones generadas.

Otro de los motivos por los que se eligió esta tecnología es porque tiene infinidad de librerías que hace más rápido su desarrollo en base a las necesidades de resolución de este problema, como lo es Hibernate, Jackson y Multi-Thread.

Jackson se utilizó para mapear los objetos provenientes del JSON a objetos, los cuales luego serán tratados para generar los escenarios necesarios.

Esta generación de escenarios estará dada por la formula NxN(poner la fórmula de la productoria), donde el estado inicial será la simulación generada desde Simugan y el conjunto resultante serán las hojas del árbol eneario generado por cada una de las generaciones.

En las diferentes pruebas, se comprobó que en ciertos casos, donde el usuario ingresa un gran volumen de variaciones en todas sus posibilidades, se necesitó paralelizar el proceso de los escenarios dado su complejidad. En este punto, se hizo uso de Pool de Thread que provee Java, para su proceso asíncrono. También, se decidió utilizar esta estrategia a partir del cuarto nivel de generación de los escenarios, ya que hasta el nivel tres, sin paralelizar la generación, el sistema se encuentra estable ya que como máximo se pueden generar 20ˆ3 generaciones (8000 escenarios), donde 20 es el número máximo de variaciones por opción que se le permiten al usuario. A partir del cuarto nivel, se observó cierta complejidad en la generación de escenarios, por lo que se decidió paralelizar la generación, aprovechando la máxima capacidad de procesamiento del servidor.