Patrones de diseño utilizados:

-Patrón arquitectónico VIPER: Aunque el patrón VIPER está más orientado a proyectos grandes, se ha utilizado en esta aplicación para demostrar su forma de uso.

Con este patrón se obtienen bastantes ventajas. Algunas de ellas:

- Se obtiene un código bastante estructurado, modulado, y limpio.
- Se aplica el principio de responsabilidad única que dicta los principios "SOLID". Cada componente que compone VIPER (Vista - Interactor - Presenter - Entiti - Router) tiene una responsabilidad clara y única, lo que también facilita a su vez, que distintos desarrolladores puedan trabajar en paralelo en el mismo proyecto.
- No se depende de implementaciones, si no de interfaces.
- Hace el código escalable. Es fácil añadir nuevas funcionalidades.
- Facilidad para implementar test, y aplicar TDD.
- Facilidad para aislar y localizar bugs en el código.

-Patrón delegado: Este patrón es utilizado cuando hay una relación de dependencia-comunicación entre dos clases, donde la clase delegada espera una respuesta de la que depende de la otra clase. Normalmente es utilizado cuando la clase delegada necesita información que se obtiene de manera asíncrona.

En este proyecto por ejemplo se ha utilizado para comunicar al "interactor" con el "presenter" en el módulo "NewList", ya que el método para obtener la lista de noticias en asíncrono en el "interactor" (hay que obtener las noticias mediante una llamada soap y luego parsear el xml obtenido). Por tanto, una vez que el "interactor" obtiene la lista de noticias ya parseas, se lo comunica a su delegado, el "presenter", el cual envía la lista de noticias en última instancia a la vista para que sea mostrada al usuario.

Frameworks utilizados:

No ha sido necesario utilizar frameworks de terceros para esta aplicación. Aunque como alternativa se podía haber utilizado Alamofire + (SWXMLHash ó SwiftyXML) para facilitar el "parseo" del xml.