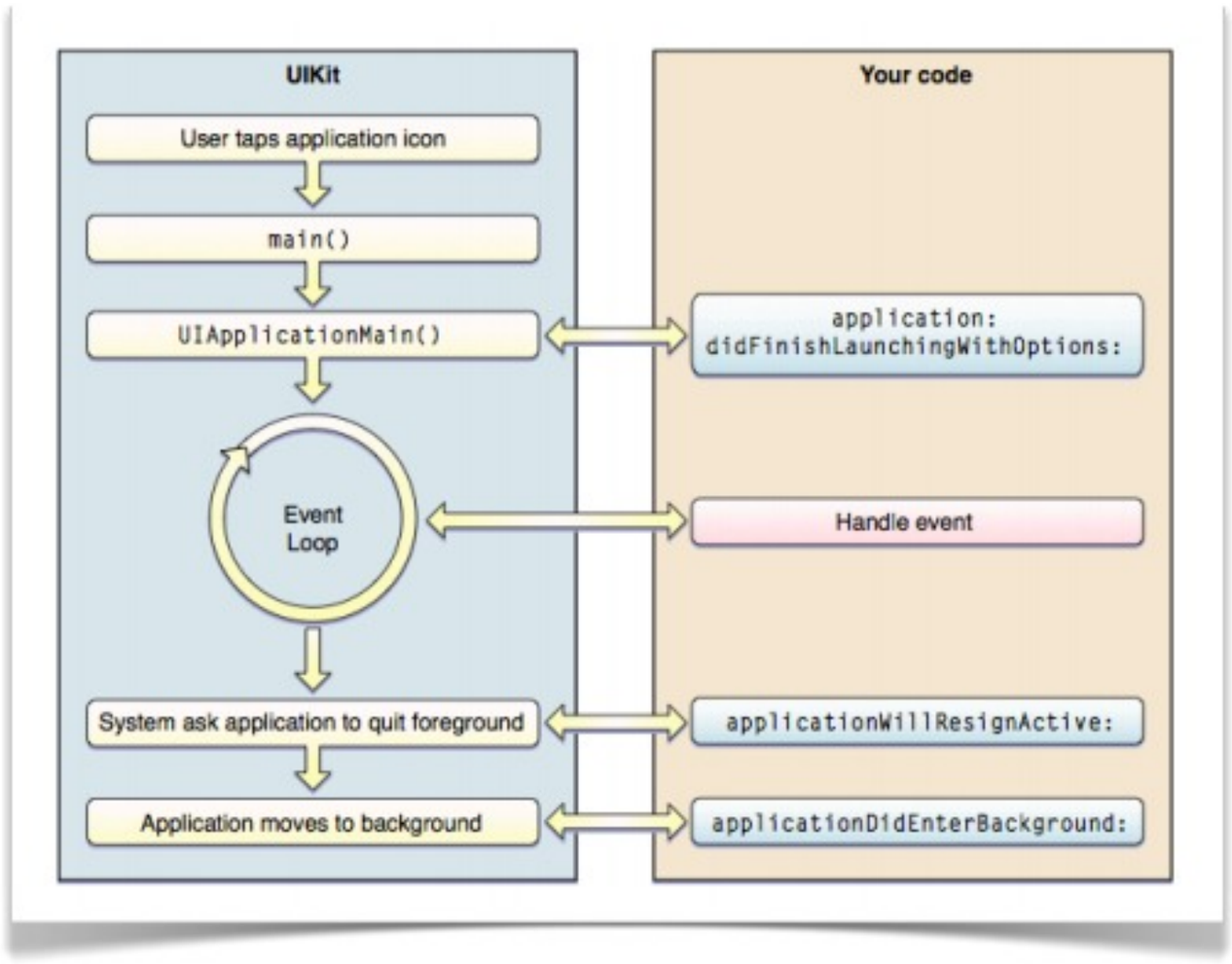
Tema 3

Ciclo de vida, Patrones de diseño y View Controllers

Ciclo de vida

Ciclo de vida



Ciclo de vida

• El primer paso del ciclo de vida se produce cuando el usuario pulsa sobre el icono de la aplicación. Al realizarse esto, se ejecuta el método principal llamado **main()**.

• En este método se hace una llamada a **UIApplicationMain()**, encargado de crear un ejecutable de la aplicación.

• Una vez finalizan estos métodos, se avisa al delegado de la aplicación, Application Delegate, mediante una llamada a **application:didFinishLaunchingWithOptions:**. En este método ya podemos escribir parte de nuestro código en caso necesario.

Ciclo de vida

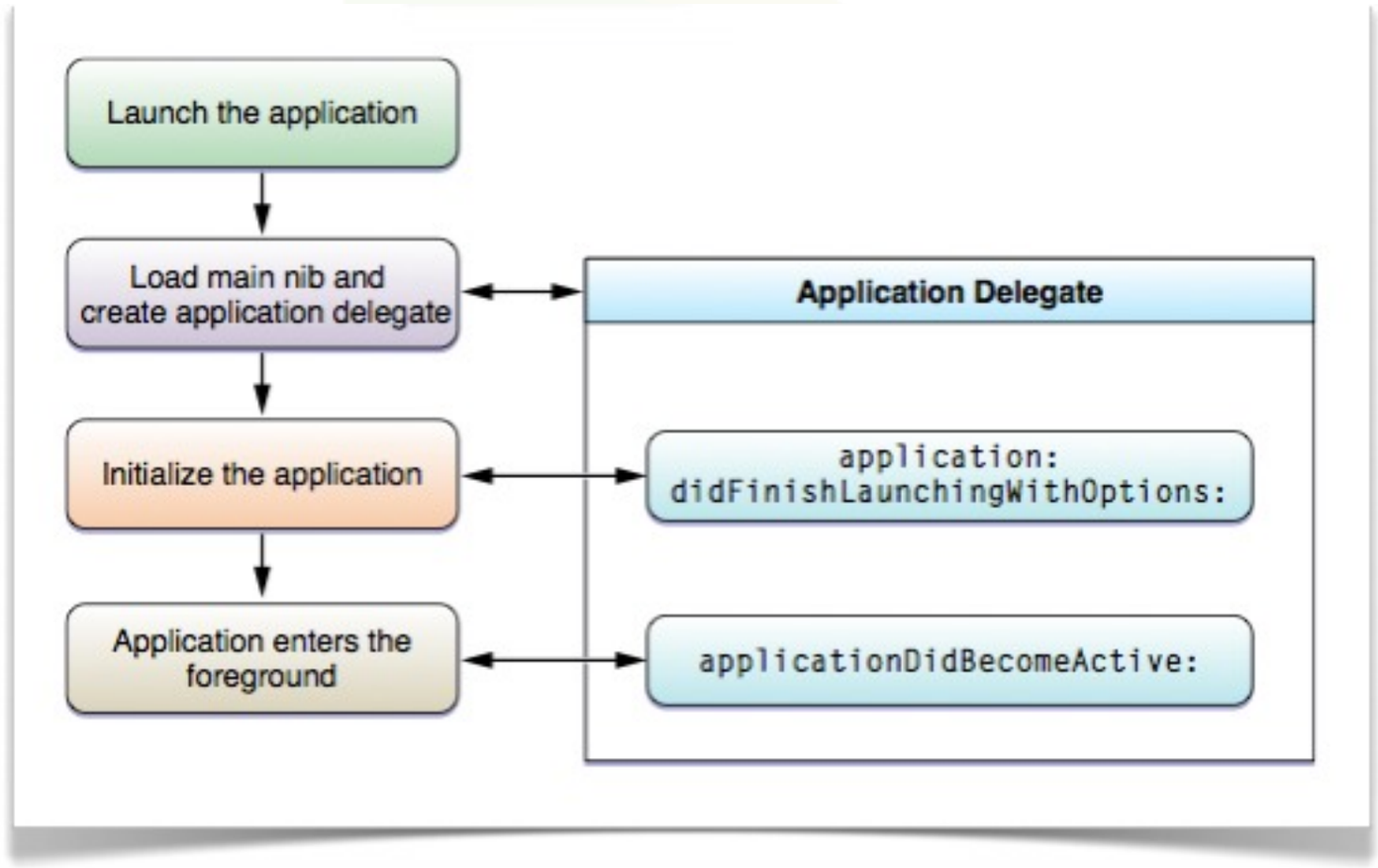
• El siguiente paso consiste en el **bucle principal** de la aplicación, que se ejecutará hasta que se decida detener la aplicación.

• En este bucle se responde a las peticiones del usuario, es decir, se captura la interacción del usuario con la aplicación para poder ejecutar el código correspondiente.

•Por último, al detener la aplicación, esta no se elimina, si no que se envía a segundo plano. Durante este proceso se avisa a la aplicación mediante el método delegado **applicationWillResignActive:** al comenzar el proceso, y mediante **applicationDidEnterBackground:** una vez se ha terminado y la aplicación se encuentra en segundo plano.

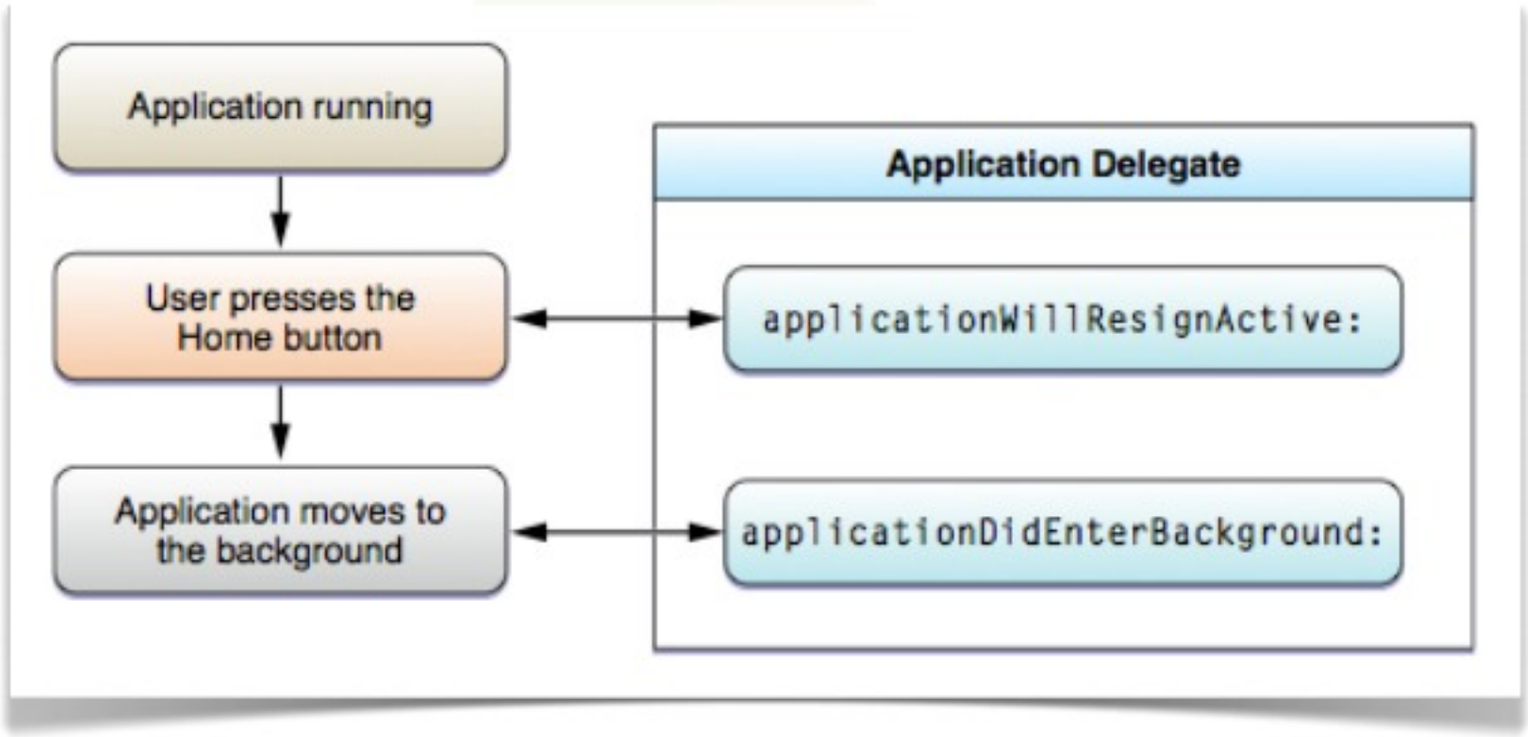
Ciclo de vida

- Inicialización de la aplicación



Introducción a iOS

- Envío de la aplicación a background



Ciclo de vida: Application Delegate

• Durante la inicialización de la aplicación se ha comentado que se avisa al delegado mediante una serie de métodos de la finalización o inicio de una serie de acciones. El delegado de la aplicación es el conocido como **Application Delegate**.

• En los proyectos que creemos recibe el nombre de ***NombreDelProyecto*AppDelegate**, tanto el fichero de la cabecera como el de la implementación. Si el proyecto se llama Project, recibirá el nombre ProjectAppDelegate.

• Estos ficheros contiene el código que se debe ejecutar cuando la aplicación termina de cargarse, cuando va a entrar en segundo plano, cuando va a volver a ser activa, etc.

Ciclo de vida: Application Delegate

- En el Application Delegate se inicia la estructura básica de la aplicación, así como la interfaz principal.

- En estos ficheros se debe incluir el código que se necesite ejecutar cuando la aplicación termine de lanzarse, finalice, se envíe a segundo plano o vuelva a estar activa.

Patrones de diseño

Patrones de diseño

- Existen tres patrones de diseño fundamentales a la hora de desarrollar una aplicación iOS.

- Estos tres patrones son los siguientes:

  - **Patrón Delegate**
  - **Patrón MVC** (Model-View-Controller)
  - **Patrón Target-Action**

Patrón Delegate

Patrón Delegate

• **Delegación** es un patrón que relaciona dos objetos de forma que uno delega en el otro. En otras palabras, uno de los objetos delega una acción a otro objeto. Este objeto que realiza la acción se conoce como **delegate** o delegado.

• Las tablas por ejemplo, utilizan este patrón para que otro objeto les informe del número de filas o secciones que deben tener.

Patrón MVC

Patrón MVC

- En el patrón MVC dividimos la estructura de la aplicación en **tres capas.**

- La primera capa es el **modelo.** Engloba las clases que utilizamos en nuestra aplicación y encapsulan los datos que se utilizan.

- La siguiente capa es la **vista**. Se corresponde con la interfaz gráfica que se muestra al usuario y contiene ventanas, controles y otros elementos con los que el usuario es capaz de interactuar.

- Por último, el **controlador** se corresponde con la lógica de la aplicación, une modelo y vista y controla la interacción que realiza el usuario con la aplicación.

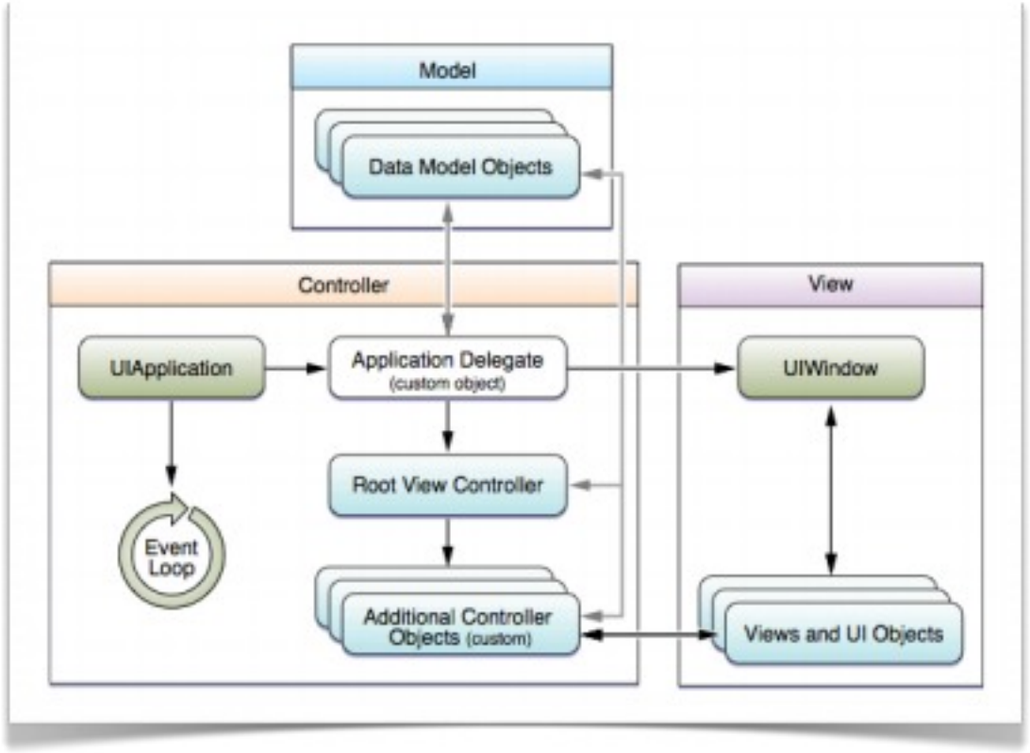
Patrón MVC

• Por norma general, el controlador es la única capa que interactúa con las otras dos de forma que el modelo y la vista únicamente se comunican mediante el controlador.

• Existe la posibilidad de que un controlador maneje varias vistas, es decir, controle y responda a eventos de más de una vista.

• Un ejemplo de este caso sería el desarrollo de una aplicación universal para iPad y iPhone. En esta situación, la interfaz gráfica sería distinta debido a los tamaños de pantalla de los dispositivos, pero la lógica interna de la aplicación es la misma. Para solucionar esto, se dispondría de dos vistas, una para iPad y otra para iPhone pero de un único controlador que se encargaría de gestionar ambas vistas.

Patrón MVC



Patrón Target-Action

Patrón Target-Action

• El patrón **Target-Action** se utiliza en el desarrollo de aplicaciones iOS para controlar y responder a los eventos producidos por el usuarios y a sus interacciones con la vista.

• Si queremos controlar y responder al pulsado de un botón asignaremos una action a dicho botón. Por otra parte, el target que indiquemos será el encargado de implementar el código necesario para capturar el pulsado del botón.

View Controllers

View Controllers

• Como ya se ha comentado, las aplicaciones iOS hacen uso del patrón Model-View-Controller. Para facilitar la implementación de este patrón existen unos controladores genéricos llamados **View Controllers**.

• Los View Controllers corresponden al **controlador** y permiten comunicarse con la vista y el modelo de forma sencilla.

• Estos controladores permiten organizar los proyectos de forma intuitiva, ya que cada controlador gestiona una vista por norma general y por tanto la división es clara y sencilla.

View Controllers

- En Objective-C los controladores son objetos que heredan de la clase **UIViewController**.

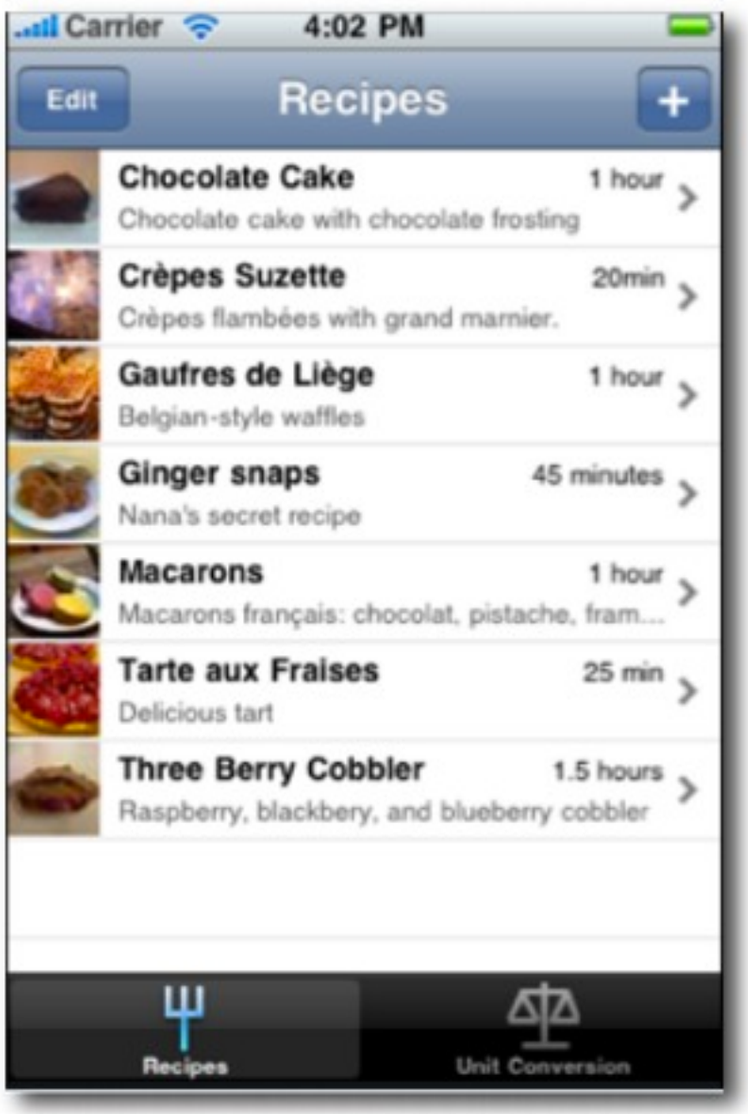
- Existen dos tipos de controladores:
  - **Custom View Controller**
  - **Container View Controller**

View Controllers - Custom View Controller

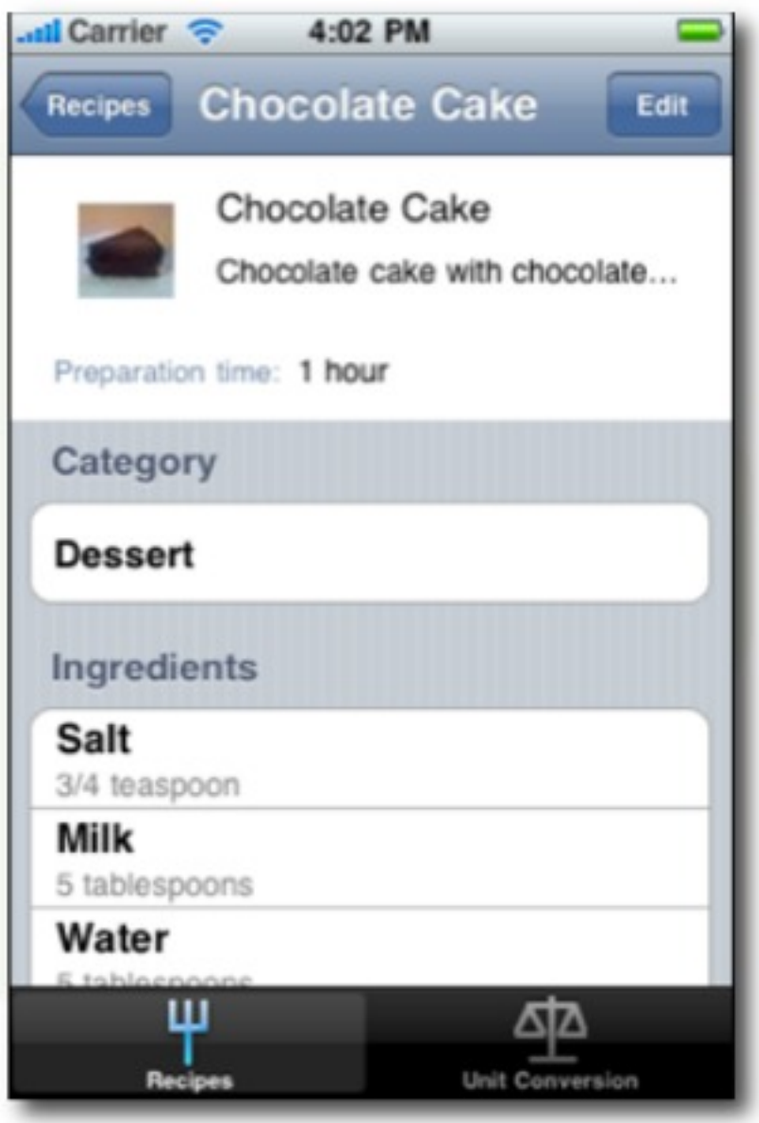
- Este tipo de controlador tiene el propósito de **mostrar contenido específico** en la pantalla.

- Ejemplos del uso de estos controladores sería mostrar una lista de elementos o las propiedades de cada uno de los elementos.

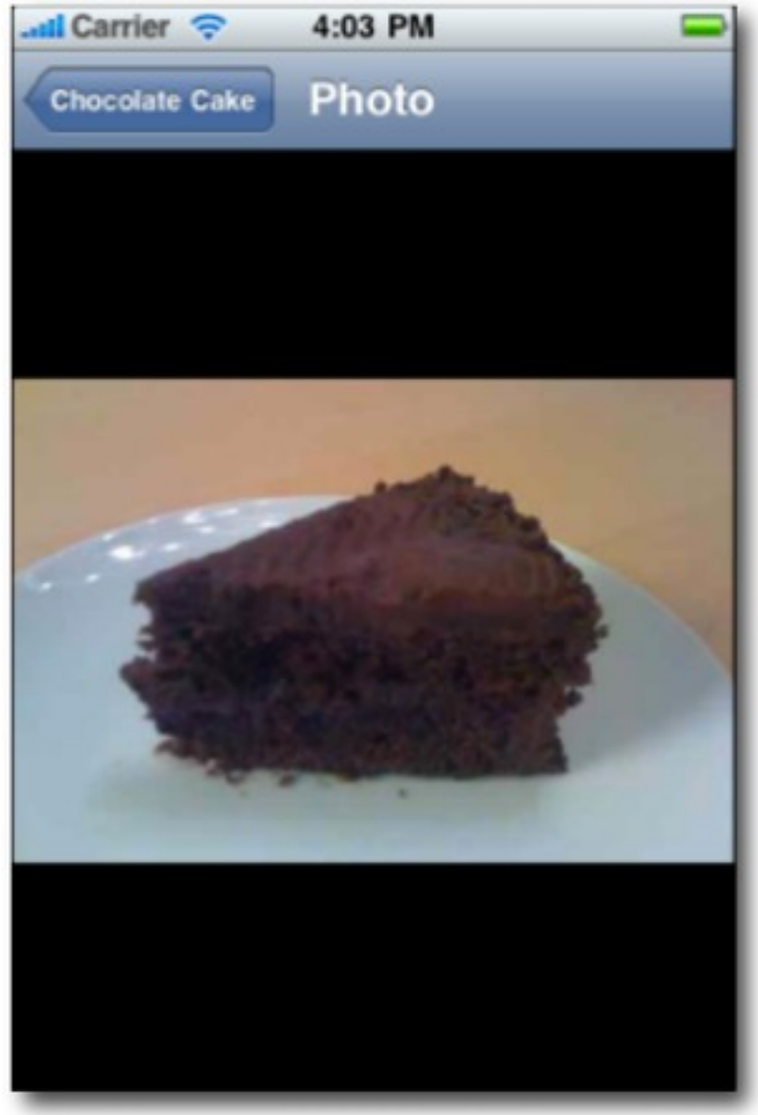
View Controllers - Custom View Controller



Lista de items



Propiedades de un ítem



Presentación de un ítem

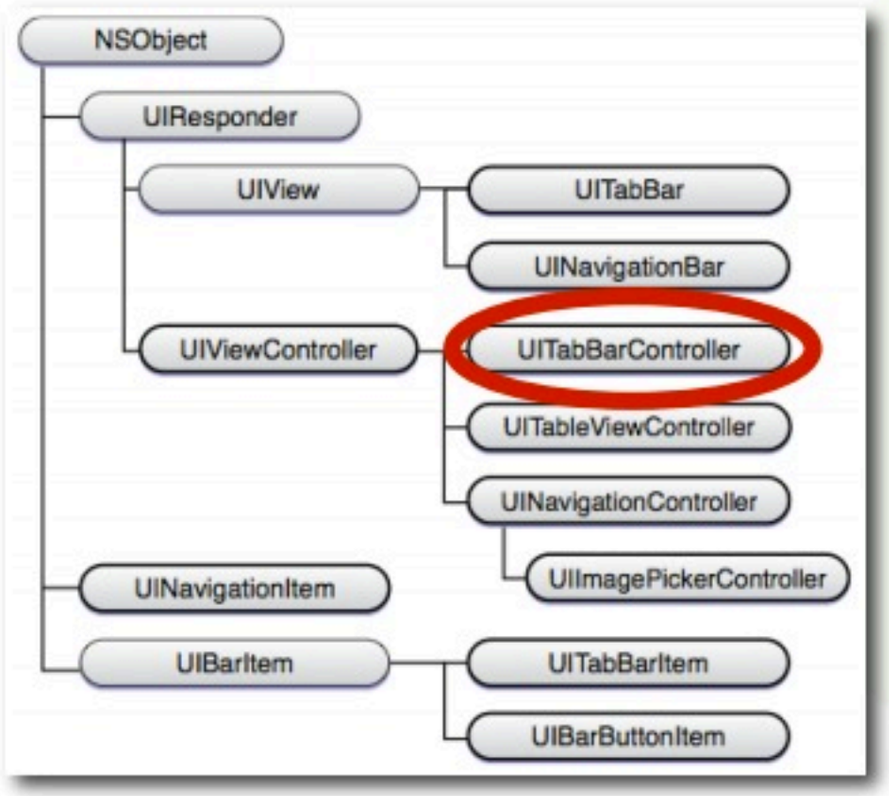
View Controllers - Container View Controller

- Los Container View Controller tienen el propósito de **gestionar otros controladores** y definir relaciones de navegación entre ellos.

- Estos controladores no se programan y vienen **predefinidos.**

- Un ejemplo de uso de este tipo de controlador sería el encargado de gestionar varias pestañas o páginas en una aplicación, conocido como Tab Bar Controller.

View Controllers - Container View Controller



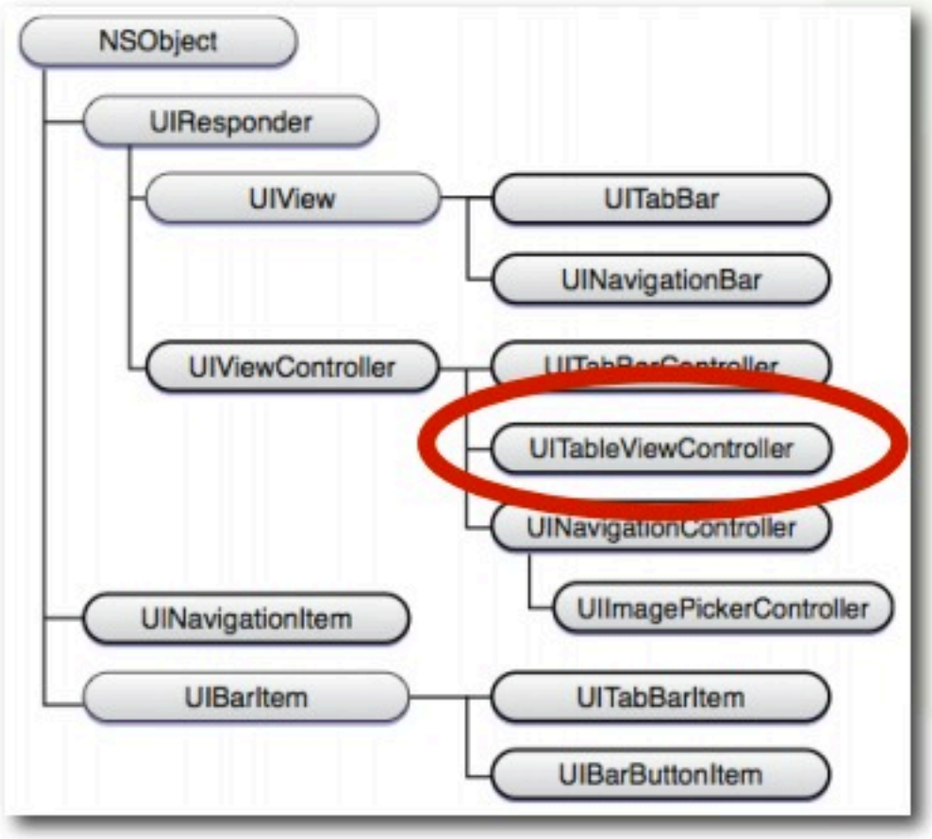
Modo Timer

Modo Stopwatch

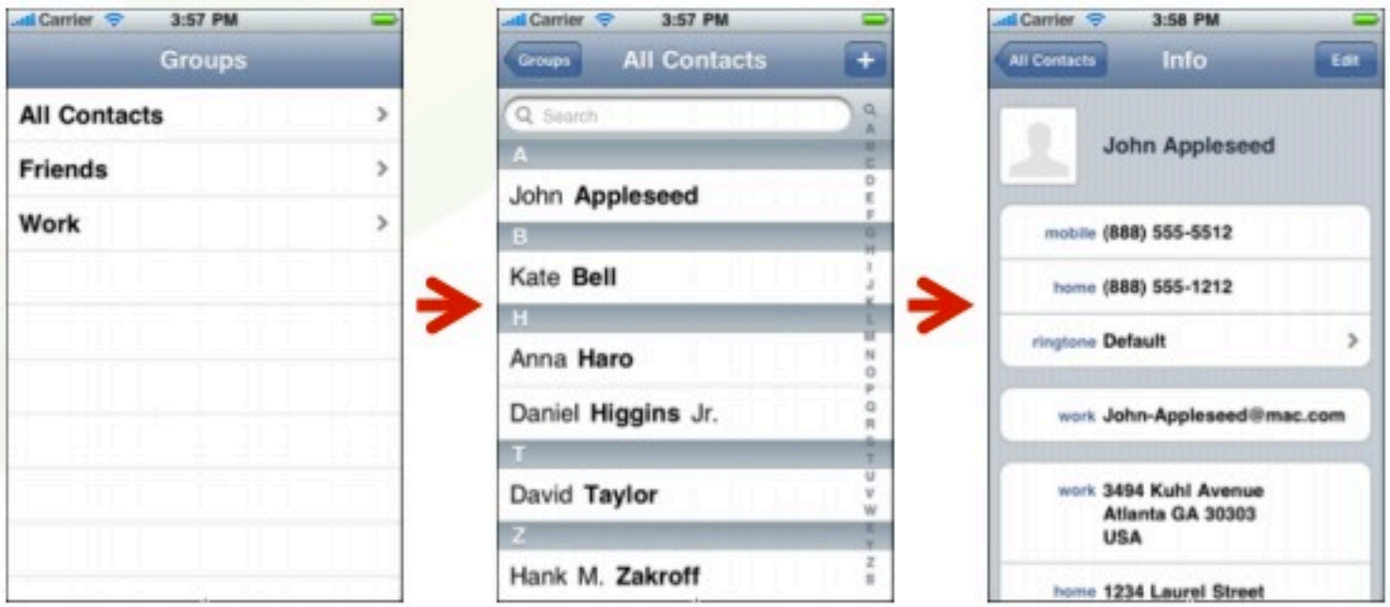
Modo World Clock

- *UITabBarController*: Sirve para dividir las aplicaciones en distintos modos de operación.

View Controllers - Container View Controller



- *UINavigationController*: información presentada de forma jerárquica.



View Controllers - Modal View Controller

- Existe una forma especial de mostrar los controladores llamada **modal.**

- Un View Controller modal **no es una subclase** de View Controller, es una forma de presentar un controlador.

- Cualquier controlador puede ser presentado de forma modal.

- Se utilizan principalmente cuando debemos salirnos del flujo de la aplicación para realizar una acción, por ejemplo, a la hora de tomar una foto debemos realizar una serie de pasos que no pertenecen a nuestra aplicación ya que usamos componentes ya creados y por tanto se muestran de forma modal.

- Otro ejemplo sería el uso de menús de selección en los que el usuario debe tomar una decisión específica o a la hora de mostrar alertas al usuario.

View Controllers - Modal View Controller