DILTest Part #1

Contact details:
Name: Ahd Radwan
Email: radwanahd@gmail.com

Code review

There are a list of changes needed to be done to improve the code quality and readability, maintainable, performance, UX, and alignment with iOS development best practice, I listed most of them here:

Naming Conventions:

- Follow the Swift naming convention to name classes and variables, and use descriptive names, use Pascal case to declare classes and types and Camel cass to declare variables and functions, for example
    - **someVC** should be **SomeViewController**
    - **closeshowDetails** should be **closeShowDetails**
- Use the same naming style with meaning full names,
    - **showDetail** & **closeshowDetails** are both relative functions with different naming styles one has “Details” the other has “Detail” in its name, they can be updated to **showDetails** & **hideDetails**

Code Organization:

Organize the code by:

- Keeping the relevant code elements grouped together, this can be achieved by implementing delegate using extensions, and making the delegate method separated from other code.
  As follows:

```
extension SomeViewController: UICollectionViewDelegate {
   // UICollectionViewDelegate methods
}
extension SomeViewController: UICollectionViewDataSource {
   // UICollectionViewDataSource methods
}
```

- Move the isIphone() method to a Utility class, e.g. DeviceUtils, to add any future relevant functions in it.

Improve Data Requests

From URLSession.shared.dataTask competition block

- Check if the response **data** object is not null, and returned with the expected data type.
- Check the error object, and show a friendly data failure UI on failure.
- Decode the Data object to the expected Data Type objects instead of casting it to [Any].
- Use weak reference instead of strong reference of **self** to avoid retain cycle.
- Update UI only on the main thread.
- The improved implementation will looks like this:

```
URLSession.shared.dataTask(with: url) { [weak self] (data, response, error) in
    if let data = data {
        do {
            let decodedData = try JSONDecoder().decode([SomeOjbect].self, from: data)
            self?.dataArray = decodedData
            DispatchQueue.main.async {
                self?.collectionView.reloadData()
            }
        } catch {
            print("Error occurred : \(error)")
        }
    }
}
```

Avoid Force Unwrapping

Avoid using force-unwrapping to unwrap optional variables, and use **guard let** to check the nullable variables and early exit.

For example:

Check if the url is not null using the following guard statement.

```
guard let url = URL(string: "testreq") else { return }
```

Threading

Only update the UI from main thread, on compilation of API, update the collectionView from the main thread, as follows:

```
DispatchQueue.main.async {
    self?.collectionView.reloadData()
}
```

Do not access a strong reference of **self** in closure, use weak reference to avoid retain cycle, as follows:

```
URLSession.shared.dataTask(with: url) { [weak self] (data, response, error) in
```

UICollecrionView

- In **cellForItemAt** return the reusable cell accessed by **dequeueReusableCell** method, instead of creating cellistance from **UICollectionViewCell**(), as follows:
  ```
  let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "someCell", for: indexPath) as?
  SomeCell
  ```
- Pass the data object to the cell, or update the cell UI components with the associated data object, before returning it.

Dealing with UIViewController

- In **viewWillAppear(_ animated:)** method the super.viewWillAppear(_ animated:) should be called. To avoid unexpected UI behavior. Currently the implemented viewWillAppear doesn't call the super method.

- When adding or removing a child view controller, the parent controller should be notified of the change, currently in the implemented code, the closeshowDetails function removes detailVC from the parent, but the showDetail function doesn't set the detailVC as a child. This can updated as follows

  //In ShowDetails function
  ```
   // First, add the view of the child to the view of the parent
   self?.detailView.addSubview(self?.detailVC.view)
  // Then, add the child to the parent
  self?.addChild(self?.detailVC)
  // Finally, notify the child that it was moved to a parent
  self?.detailVC.didMove(toParent: parent)
  ```

  //In closeshowDetails function
  ```
  // First, notify the child that it's about to be removed
  self?.detailVC?.willMove(toParent: nil)
  // Then, remove the child from its parent
  self?.detailVC?.removeFromParent()
  // Finally, remove the child's view from the parent's
  self?.detailVC?.view.removeFromSuperview()
  ```

- Another mistake in adding and removing the child view, is that the showDetail function adds the detailView to the parent controller view "self.view.addSubview(self.detailView)" while it should add the detailsVC view to the container view which is the detailView, this

mistake fixed in the first line of the code in the previous point.
“`self?.detailView.addSubview(self?.detailVC.view)`”

Readability

- Avoid Magic Numbers and Strings: Replace magic numbers and strings with named constants or enums. To improve code readability by providing context to the values used in the code.
  Here is the list of the Number and strings that should be declared in named constants.
  - Animation duration
  - Reusable cell identifier `"someCell"`
  - Dimensions, as in “`detailViewWidthConstraint.constant = 100`”
  - The width multiplier (`0.2929`) is used twice, once declared as a constant called **widthMultiplier** and the second as a number in an equation in minimumLineSpacingForSectionAt method.
  - The “`frameWidth = (view.frame.width * 0.2929 * 3) + 84`” equation has a list of undeclared numbers, it's not clear what these numbers are. All these numbers should be replaced with named constants.

- Add more comments to describe measurement equations especially in minimumLineSpacingForSectionAt method.

Friendly UX

- Show a spinner, or user friendly UI that tells the user there is data being loaded.
- Handle data failure, tell the user that the data failed to load, and show a descriptive error message.
- Allow the user to refresh the data, if the data failed to load.
- Show friendly UI display an empty layout, in case the response data is empty.

Others

- The **dissmissController()** method is empty, and there is no comment, or TODO tag, to describe what will be implemented here.
- Remove unused Outlets, as in collectionViewTopConstraint.

Typos:

There is a typo in **dissmissController**, update it to **dismissController**

1. Feedback

Naming:
Consider using more descriptive names for classes, methods, and variables to make the code more readable and self-explanatory.

Network Request Handling:
Instead of force-casting, use a result type or error handling when decoding network response data.

UICollectionView Methods:
Implement cellForItemAt to return the reusable cells instead of a generic UICollectionViewCell.

Code Organization:
Group related code together and use extensions to separate protocol conformance for better organization. And use "// MARK: -" annotation to organize related code together.

Friendly UX:
Always let the users be aware that there is a data being loaded, and provide them with data loading results.

Typos:
Use Xcode editor or other tools to check typos in your code, to provide more readable code.

Magic Numbers and Strings
Always define the number and strings, you use in well defined named constants, to improve readability and maintainability of your code.

Memory management:
Never use strong reference of self in closures, instead use [weak self].
Update UI components only from the main thread.

UIViewController:
Follow iOS development best practice to deal with UIViewController and other UI components, you need to notify the controller when adding or removing a child controller, always call super methods in UIViewController lifecycle methods.