NoteManagementController.swift

```
/// Represents each of the user's folders contains only a label for the folder name
class DirectoryCollectionCell: UICollectionViewCell {
       /// sets the  cell's name to the name of the folder
       /// - Parameter name: String name of the folder
       func displayContent(name: String) {}
}

/// The view controller for the splash page which includes collection view
class NoteManagementController: UIViewController, UICollectionViewDelegate,
UICollectionViewDataSource {
        ///overrides function so the bucket handler is initialized and the names of
the directories are received
    override func viewDidLoad() {}

    ///sets the datasource after viewDidLoad runs
    /// - Parameter animated: if we want to animate the display
    override func viewDidAppear(_ animated: Bool) { }

    ///Tells the controller how many cells to display
    /// - Parameters:
    ///   - collectionView: UICollectionView the collection view on the controller
    ///   - section: Int the number of sections in the controller
    /// - Return: Int: the number of cells needed in the view
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
section: Int) -> Int { }

    ///sets content of each cell from data array and styles cells
    /// - Parameters:
    ///   - collectionView: UICollectionView the collection view on the controller
    ///   - indexPath: IndexPath gets the cell that we want to modify
    /// - Return: UICollectionViewCell: the formatted collection view cell to add to
the view
    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell { }

}
```

AWSBucketHandler.swift

```
/// AWSBucketHandler makes all requests to the S3 bucket using the configuration
specified
/// This class is instantiated and passed through many view controllers
class AWSBucketHandler {

    ///initializes and configures credential provider with userpool id and credentials
    /// - Parameter id: a unique identifier for the logged in user, also the name of
their folder in the S3 bucket
    init(id: String) { }
```

```
    ///gets the names of all files in the user's directory and stores it in a global
variable
    /// - Parameter completion: this is an event callback that lets the caller execute
some function after the request completes
    func getAllFiles(completion: @escaping (AnyObject?)->()) {  }

    //isolates directory names and returns to caller, ensures non-redundancy
    /// Extracts the list of directories from the file names to populate the splash
page of user folders
    public func getDirectories() -> Array<String> { }
}
```

AlteredImage.swift

```
/*
Abstract:
An object that augments a rectangular shape that exists in the physical environment.
*/

/// - Tag: AlteredImage
class AlteredImage {

    /// A delegate to tell when image tracking fails.
    weak var delegate: AlteredImageDelegate?


    /// Displays the altered image using the anchor and node provided by ARKit.
    /// - Tag: AddVisualizationNode
    /// - Parameters:
    ///   - anchor: augmented reality anchor
    ///   - node: scene node
    func add(_ anchor: ARAnchor, node: SCNNode) {}

    /**
     If an image the app was tracking is no longer tracked for a given amount of time,
invalidate
     the current image tracking session. This, in turn, enables Vision to start
looking for a new
     rectangular shape in the camera feed.
     - Tag: AnchorWasUpdated
     - Parameter anchor: anchor to update to
     */
    func update(_ anchor: ARAnchor) { }


    /// Prevents the image tracking timeout from expiring.
    private func resetImageTrackingTimeout() {}
```

```
    /// Alters the image's appearance by applying the "StyleTransfer" Core ML model to
it.
    /// - Tag: CreateAlteredImage
    func createAlteredImage() {}

/**
 Tells a delegate when image tracking failed.
  In this case, the delegate is the view controller.
 */
protocol AlteredImageDelegate: class {}
```

RectangleDetector.swift

```
/*
Abstract:
An object that checks images for rectangles.
*/

/// - Tag: RectangleDetector
class RectangleDetector {

    /// Search for rectangles in the camera's pixel buffer,
    ///  if a search is not already running.
    /// - Tag: SerializeVision
    /// - Parameter pixelBuffer: pixel buffer of current user environment
       private func search(in pixelBuffer: CVPixelBuffer) {}

    /// Check for a rectangle result.
    /// If one is found, crop the camera image and correct its perspective.
    /// - Tag: CropCameraImage
    /// - Parameters:
    ///   - request: vision request
    ///   - error: error
    private func completedVisionRequest(_ request: VNRequest?, error: Error?) {}
```

Utilities.swift

```
/*
Abstract:
The sample app's reusable helper functions.
*/


extension CIImage {

    /// Returns a pixel buffer of the image's current contents.
    /// - Parameter pixelFormat: Format of pixel
    func toPixelBuffer(pixelFormat: OSType) -> CVPixelBuffer? {}

    /// Returns a copy of this image scaled to the argument size.
    /// - Parameter size: the size of the image
```

```
    func resize(to size: CGSize) -> CIImage? { }

extension CVPixelBuffer {
       /// Returns a Core Graphics image from the pixel buffer's current contents.
       func toCGImage() -> CGImage? { }

       /// Creates a SceneKit node with plane geometry, to the argument size,
       rotation, and material contents.
       /// - Parameters:
       ///   - size: size of image
       ///   - rotation: rotation of image
       func createPlaneNode(size: CGSize, rotation: Float, contents: Any?) -> SCNNode
       {}
}
```

Visualization.swift

```
/*
Abstract:
A SceneKit node that fades between two images.
*/

class VisualizationNode: SCNNode {

    /**
     Create a plane geometry for the current and previous altered images sized to the
argument
     size, and initialize them with transparent material. Because `SCNPlane` is
defined in the
     XY-plane, but `ARImageAnchor` is defined in the XZ plane, you rotate by 90
degrees to match.
     */
    init(_ size: CGSize) { }

    /// Assigns a new current image and fades from the previous image to it.
    /// - Tag: ImageFade
    /// - Parameter alteredImage: image to display in the node
    func display(_ alteredImage: CGImage) {}
}
```

ARAnnotationViewController.swift

```
/*
Abstract:
A view controller that recognizes and tracks images found in the user's environment.
*/

class ARAnnotationViewController: UIViewController {

    /// Runs image tracking session
```

```
    /// - Parameters:
    ///   - trackingImages: A set of AR Reference images to track
    ///   - runOptions: Options for the AR session
    private func runImageTrackingSession(with trackingImages: Set<ARReferenceImage>,
                                         runOptions: ARSession.RunOptions =
[.removeExistingAnchors]) {}


    /// Shows message on screen when rectangle is not found
    /// - Parameters:
    ///   - message: The message to display
    ///   - autoHide: by default message is hidden
    func showMessage(_ message: String, autoHide: Bool = true) {}

    /// Set message as hidden
    /// - Parameter hide: boolean
    private func setMessageHidden(_ hide: Bool) {
        DispatchQueue.main.async { }
 }

extension ARAnnotationViewController: ARSCNViewDelegate {

    /// - Tag: ImageWasRecognized
    /// Adds image to node tree so that it can be rendered, sets message to be hidden
as rectangle is found
    /// - Parameters:
    ///   - renderer: scene renderer
    ///   - node: scene node to add anchor
    ///   - anchor: ar anchor for the image that should be added
    func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor:
ARAnchor) { }

    /// - Tag: DidUpdateAnchor
    /// To update anchor when image is moved
    /// - Parameters:
    ///   - renderer: scene renderer
    ///   - node: scene node to add anchor
    ///   - anchor: ar anchor for the image that should be added
    func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for anchor:
ARAnchor) { }

    /// When errors in session
    /// - Parameters:
    ///   - session: AR session
    ///   - error: The error
    func session(_ session: ARSession, didFailWithError error: Error) { }

extension ARAnnotationViewController: RectangleDetectorDelegate {
    /// Called when the app recognized a rectangular shape in the user's environment.
    /// - Tag: CreateReferenceImage
```

```
    /// - Parameter rectangleContent: the image corresponding to the content within
rectangle detected
    func rectangleFound(rectangleContent: CIImage) {}

/// Enables the app to create a new image from any rectangular shapes that may exist
in the user's environment.
extension ARAnnotationViewController: AlteredImageDelegate {
    /// When tracking is lost, so that a new image is tracked
    /// - Parameter alteredImage: the image that is no longer being tracked
    func alteredImageLostTracking(_ alteredImage: AlteredImage) {}
}
```

Handwriting Handler.swift

```
/* Given an image, parses it for words and stores in a global dictionary for the given
user, tracking what pages the words appear on */
class HandwritingManager{

func processImage(in uiImage: UIImage, pgNum: Int){}
// uiImage is an image that is passed into the function, pgNum is a page number
```

ImageCaptureViewController.swift

```
/*
Abstract:
A view controller that allows users to upload photos to the app and receive image
overlay codes for them.
*/

class ImageCaptureViewController: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {
    /// requests user's permission to access their camera
    func requestCameraPermission() {}

    /// opens the camera
    func presentCamera() {}

    /// alerts the user that they have not given access to the camera
    func alertCameraAccessNeeded() {}

    /// takes the image passed from either the camera or the Photo Library,
    /// generates a code for it, and stores it according to that code
    /// - Parameters:
    ///  - picker: source that picked the image
    ///  - info: photo data
    func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {}

    /// stores image and associates it with a code
    /// - Parameters:
    ///  - image: image to be stored
```

```
    ///  - key: code to associate with the image
    ///  - storageType: type of storage (filesystem or other)
    private func store(image: UIImage, forKey key: String, withStorageType
storageType: StorageType) {}

    /// retrieves image from the user's filesystem based on given code
    /// - Parameters:
    ///  - key: code associated with the desired image
    ///  - storageType: type of storage (filesystem or other)
    private func retrieveImage(forKey key: String, inStorageType storageType:
StorageType) -> UIImage? {}

    /// helper function that gets the filepath for an image based on its code
    /// - Parameter key: code associated with the desired image
    private func filePath(forKey key: String) -> URL? {}

    /// - Parameter length: the length of the random string you want to generate
    /// generates a random string of specified length to be used as a code
    func randomString(length: Int) -> String {}

}
```