Outlining the UML Structures & the Crucial Algorithms for **AustinDrinks**

AustinDrinks is built on the Model View Controller (MVC) architecture. There are five models in **AustinDrinks**. One model is the API model governed by Apple, Inc.'s software engineers. This model is tested for bugs by the Apple team(s). The other four are tested by me.

Application Structure in Six Figures

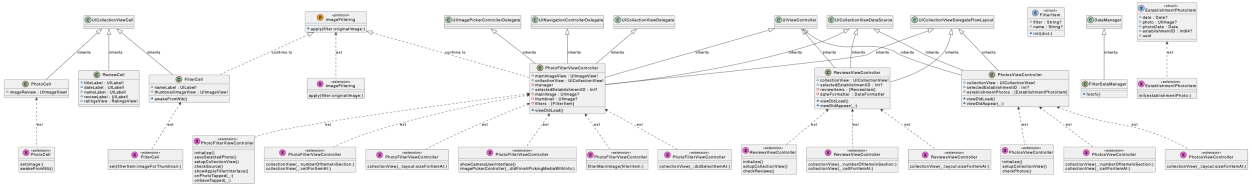

Figure 1.0 The First UML Frame—User Feedback and Establishment Experience

1. *Reviews & Photos Architectural Components*: Starting from the left and reading to the right we find there are central classes: PhotoCell; ReviewCell; FilterCell; ImageFiltering; PhotoFilterViewController; ReviewsViewController; PhotosViewController; FilterDataManager; and EstablishmentViewItem.

PhotoCell, ReviewCell, and FilterCell inherit class attributes and behaviors from the Apple API class UICollectionViewCell. PhotoCell and FilterCell each have refactored extension classes. And FilterCell "confirms to", or is the interface realization of Apple's API protocol ImageFiltering. PhotoFilterViewController also confirms to the ImageFiltering protocol. Further, the ImageFiltering protocol has an extension (X) for its *apply()* method.

PhotoFilterViewController inherits from Apple's API classes UIImagePickerControllerDelegate, UINavigationControllerDelegate, UICollectionViewDelegate, UIViewController, UICollectionViewDataSource, and UICollectionViewDelegateFlowLayout. PhotoFilterViewController also has six extensions X each of which is a refactorization of the a method's use in the main class PhotoFilterViewController.

ReviewsViewController inherits from Apple's API classes UIViewController, UICollectionViewDataSource, and UICollectionViewDelegateFlowLayout. ReviewsViewController further has three extensions X wherein different methods in ReviewsViewController are refactored allowing for separation of concerns. The exact same inheritance and extension structure/analysis is embodied in the PhotosViewController.

FilterDataManager inherits from the class DataManager, mentioned above, which is maintained by Apple.

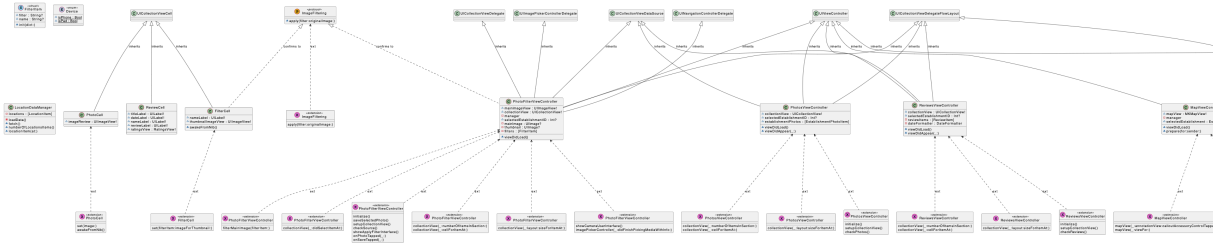

Figure 2.0 The First UML Frame—User Feedback and Establishment Experience

It's worth mentioning there is a *struct* FilterItem. It's used in the mechanics of the "Filter-. . ." classes above. There is also a EstablishmentPhotoItem *struct* which sets the attributes and API settings to be used among the rest of the

"Photo-. . ." classes. A particularly interesting API setting is the **establishmentId** = [some random integer] which in the EstablishmentPhotoItem *struct* is encoded to CoreData by use of the UUID() API call. Essentially what this does is store the User Experience Photos to CoreData which in turn stores it to the **AustinDrinks** application on the user's device, wherein it can be shared with other **AustinDrinks** users.

2. *Basic Data Management & ViewControllers:* In Figure 3.0 we find the central classes are: MapViewController; EstablishmentDetailViewController; ReviewFormViewController; LocationViewController; MapDataManager; RatingsView; NoDataView; and EstablishmentItem. There is the DataManager (It just appears here again, not as a duplicate, but as a re-worked automatically generated SwiftPlantUML artifact with more relationships in this frame.) protocol, the ReviewItem *struct,* LocationItem *struct*, and the CoreDataManager *struct.* The CoreDataManager has an extension X with a method *shared()*. This method and its CoreDataManager *struct* are related to the notion of CoreData previously discussed.

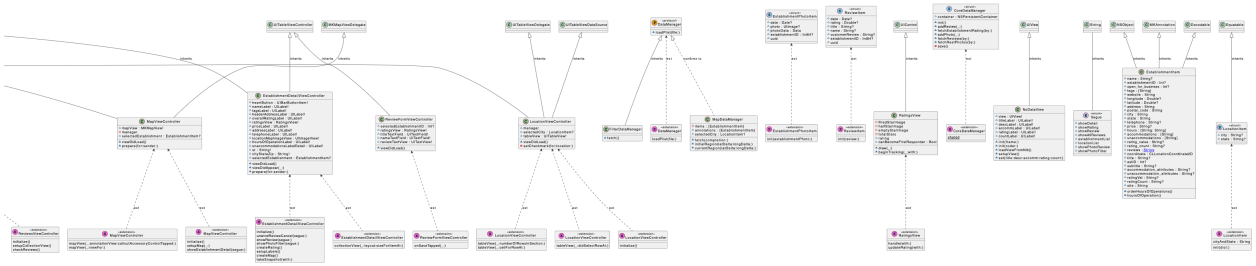

Figure 3.0 The Third UML Frame—Data Management and Establishment Detail View (Page) Mechanics

CoreDataManager is the construct (the API "glue" one might say) by which User Experience Reviews and Photos are added, saved or stored, fetched, and ultimately shared. It's a vital component to the ESTABLISHMENT DETAIL page of **AustinDrinks**.

MapViewController inherits from the Apple APIs UICollectionViewDelegateFlowLayout and MKMapViewDelegate, both Apple APIs. It also has two extensions X which refactor method use in the MapViewController.

EstablishmentDetailViewController also inherits from UICollectionViewDelegateFlowLayout, as well as the Apple APIs UITableViewController. EstablishmentDetailViewController has two extensions X where methods in the main class EstablishmentDetailViewController refactored for clarity and separation of concerns. EstablishmentDetailViewController is a mission-critical class in that its algorithms and attributes deliver the nitty-gritty of the **AustinDrinks** information value. That is, once one has Explored and discovered the establishment the user would like to find more about, the ESTABLISHMENT DETAIL page is where this class comes into play. Its attributes govern the fields which will populate the crucial establishment information which a user comes to **AustinDrinks** to find conveniently discover.

ReviewFormViewController inherits UITableViewController, complete with an extension X with a method for saving-when-tapped a user review.

LocationViewController inherits from Apple API classes UIViewController, UITableViewDelegate, and UITableViewDataSource, which can be fully seen by comparing Figure 2.0 and Figure 3.0. This class has three refactored extensions dealing respectively with LocationViewController methods pertaining to each of the API classes. This class effectively allows for the +LOCATION button to work, wherein is able to select the button +LOCATION; then one may choose from among the 11 Austin and surrounding areas, which once a selection is made a checkmark will be placed by the destination. This restricts the selection of breweries, distilleries, and wineries to a single city.

MapDataManager "confirms to", conforms to—depending on custom—the DataManager protocol. It is a key component to mapping the full set of drinking establishments with map pins to a map of Austin and the 10 other surrounding areas.

RatingsView handles and updates starred ratings on a 5-star-scale. It inherits from Apple's API class UI control. Basically, it allows for partial or full filling of stars based upon an average number of starred ratings. So, if for example, one user gave an establishment two stars and another user gave the same establishment 4 stars the stars display would show a 3 star rating.

NoDataView is the pop screen *only if* there are no JSON files either in the application or there is an improper parsing of the JSON data for some city. It inherits from UIView.

EstablishmentItem is absolutely crucial to the whole **AustinDrinks** application. Without it, **AustinDrinks** would simply be defunct, or at best useless. EstablishmentItem inherits from Apple API classes NSObject, MKAnnotations, and Decodable, and effectively speaks to and decodes each city's JSON file when selected. So, every key in the JSON is "grabbed" at this stage of operation and each key's value is then fed into appropriate classes such as, for example, the EstablishmentDetailViewController.

Finally, in Figure 3.0, there is the LocationItem *struct* which establishes the city and state, say, "Austin, TX" and aides in displaying this user's selection in the LocationViewController.

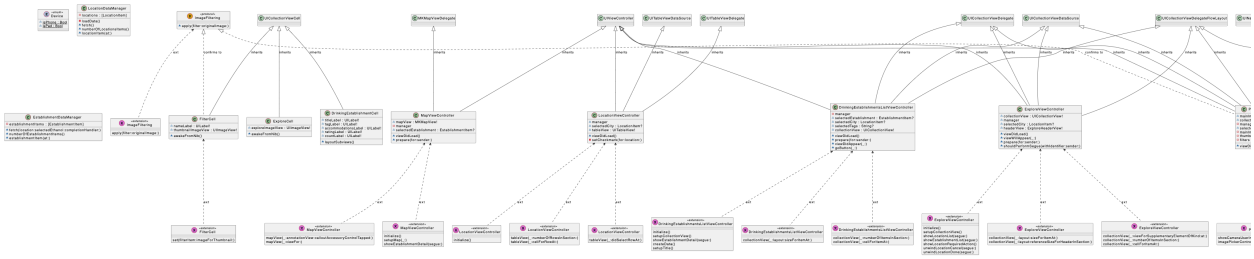

Figure 4.0 The Fourth UML Frame—The DrinkingEstablishmentListViewController and the Explore View (Page) Mechanics

The DrinkingEstablishmentListViewController class is key to working with the ExploreViewController class. These are also absolutely crucial to the working of the whole system. In fact, each aids in designing application flow. One opens the **AustinDrinks** application and finds herself on the Explore page which is governed by the ExploreViewController. From there one must select a location with the +LOCATION button. This is where LocationViewController comes into play. Upon choosing a destination, one may then select a craft-beverage type on the Explore page. Having chosen a beverage type, one is then re-directed by the ExploreViewController class to all the craft-beverages in the city of the user's choice of the chosen craft-beverage type.

ExploreViewController inherits from Apple API classes UITableViewDelegate, UICollectionViewDelegate, UICollectionViewDataSource, and UICollectionViewDelegateFlowLayout. ExploreViewController has three refactored extensions X. Each manages a different component of Explore page's layout and mechanics.

DrinkingEstablishmentListViewController inherits from the same four Apple APIs as ExploreViewController. Just as with ExploreViewController there are three extensions X, which together define the crucial drinking establishment listings page behavior. Each is a refactorization of methods utilized in the class itself.

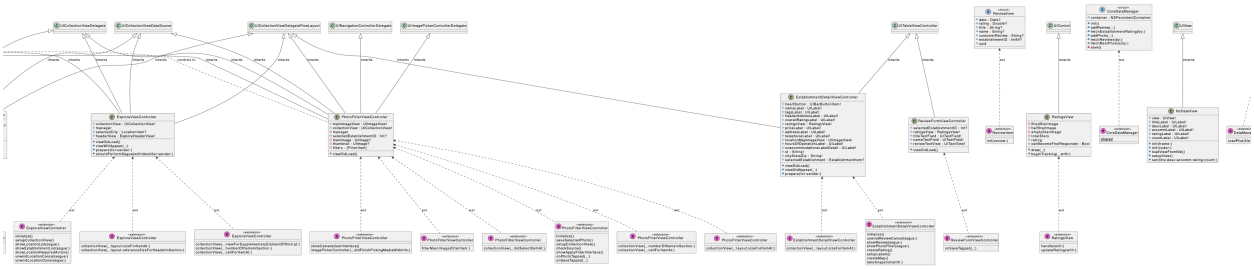

Figure 5.0 The Fifth UML Frame—The EstablishmentDetailViewController (Page)

3. *AppDelegate & SceneDelegate:* AppDelegate and SceneDelegate are automatically generated by Xcode when Xcode builds out the skeleton of a project. Basically, each allows Xcode to communicate with developer-written code and the various Storyboards built in the project, including especially the Main Storyboard. These files are vital to any project and are built deeply for the Swift API. There is a heavy amount of abstraction with these two files.

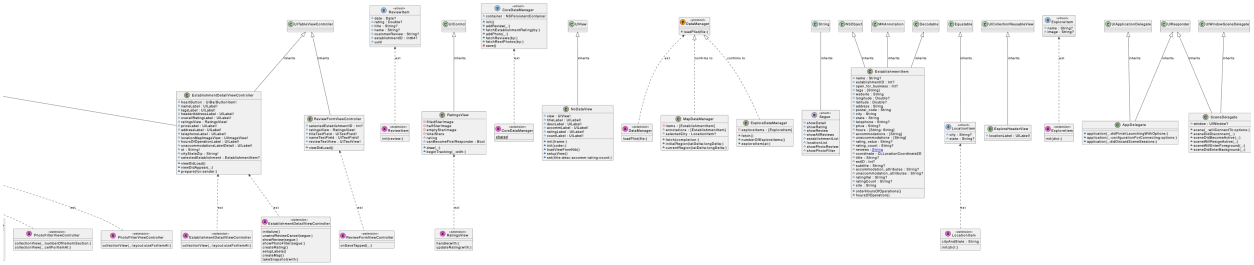

Figure 6.0 The Sixth UML Frame—Deep Apple API: AppDelegate & SceneDelegate

A Brief Overview of Key Algorithms

1. *Class EstablishmentItems:*

```
import UIKit
import MapKit

// JSON Decodable template
class EstablishmentItem: NSObject, MKAnnotation, Decodable {

    // All attributes and their types in the JSON data structure.
    let name: String?
    let establishmentID: Int?
    let open_for_business: Int?
    let tags: [String]
    let website: String
    let longitude: Double?
    let latitude: Double?
    let address: String
    let postal_code: String
    let city: String
    let state: String
    let telephone: String?
    let price: String?
    var hours: [String: String]
    let accommodations: [String]
    let unaccommodations: [String]
    let rating_value: String?
    let rating_count: String?

    enum CodingKeys: String, CodingKey {
        case name = "name"
        case establishmentID = "id"
        case open_for_business = "open_for_business"
        case tags = "tags"
        case website = "website"
        case longitude = "longitude"
        case latitude = "latitude"
        case address = "address"
        case postal_code = "postal_code"
        case city = "city"
        case state = "state"
        case telephone = "telephone"
        case price = "price"
        case hours = "hours"
        case accommodations = "accommodations"
```

```
        case unaccommodations = "unaccommodations"
        case rating_value = "rating_value"
        case rating_count = "review_count"
    }

    // Coordinate setter
    var coordinate: CLLocationCoordinate2D {
        guard let latitude = latitude, let longitude = longitude else {
            return CLLocationCoordinate2D()
        }
        return CLLocationCoordinate2D(latitude: latitude, longitude: longitude)
    }

    // Setter
    var title: String? {
        name
    }

    // Setter
    var estID: Int? {
        establishmentID
    }

    // Iterates through a list of tags, where tags in the code are understood to be
    // drinking establishment types--namely, brewery, distillery, or winery.
    var subtitle: String? {
        if tags.isEmpty {
            return ""
        } else if tags.count == 1 {
            return tags.first
        } else {
            return tags.joined(separator: ", ")
        }
    }

    // Iterates through a list of establishment accommodations
    var accommodation_attributes: String? {
        if accommodations.isEmpty {
            return ""
        } else if accommodations.count == 1 {
            return accommodations.first
        } else {
            return accommodations.joined(separator: "\n")
        }
    }

    // Decodable does not order an already unordered dictionary, so this function orders the
hours of operation from
    // the beginning of the week to the end.
    func orderHoursOfOperations() -> [[String: String]] {
        let order = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
        var list_h: [[String: String]] = [[:]]
        for day in order {
            for (key, value) in hours {
                if (key == day) {
                    var dict: [String:String] = [:]
                    dict[key] = value
                    list_h.append(dict)
                    break
                }
            }
        }
        return list_h
    }

    // Returns the ordered hours of operation
    func hoursOfOperation() -> String {
        let h = orderHoursOfOperations()
        var hoursOfOp = ""
        for obj in h {
            for (key, value) in obj{
            hoursOfOp.append("\(key) : \(value)\n")
```

```
            }
        }
        return hoursOfOp
    }

    // Iterates through a list of what I'm calling "unaccommodations" or
    // other information one might like to know that is not an accommodation.
    // An example is "No pets allowed".
    var unaccommodation_attributes: String? {
        if unaccommodations.isEmpty {
            return ""
        } else if unaccommodations.count == 1 {
            return unaccommodations.first
        } else {
            return unaccommodations.joined(separator: "\n")
        }
    }

    // Setter
    var ratingVal: String? {
        rating_value
    }

    // Setter
    var ratingCount: String? {
        rating_count
    }

    // Setter
    var site: String {
        website
    }
}
```

The above algorithm is *absolutely vital to* **AustinDrinks**. In short, it is the algorithm that both decodes data from the 11 distinct JSON files and organizes the data for use. Additionally, there are helper functions, orderHoursOfOperation() and hoursOfOperation() which together help to organize data in the unordered list of dictionaries in the 11 different JSON files.

2. Class CoreDataManager:

```
import CoreData
import Foundation

// How the Reviews and Photos are saved to the user's device for sharing
struct CoreDataManager {
    let container: NSPersistentContainer
    init() {
        container = NSPersistentContainer(name: "AustinDrinksModel")
        container.loadPersistentStores {
            (storeDesc, error) in
            error.map {
                print($0)
            }
        }
    }

    func addReview(_ reviewItem: ReviewItem) {
        let review = Review(context: container.viewContext)
        review.date = Date()
        if let reviewItemRating = reviewItem.rating {
            review.rating = reviewItemRating
        }
        review.title = reviewItem.title
        review.name = reviewItem.name
        review.customerReview = reviewItem.customerReview
        if let reviewItemEstID = reviewItem.establishmentID {
            review.establishmentID = reviewItemEstID
        }
        review.uuid = reviewItem.uuid
```

```
        save()
    }

    func fetchEstablishmentRating(by identifier: Int) -> Double {
        let reviewItems = fetchReviews(by: identifier)
        let sum = reviewItems.reduce(0, {$0 + ($1.rating ?? 0)})
        return sum / Double(reviewItems.count)
    }

    func addPhoto(_ estPhotoItem: EstablishmentPhotoItem) {
        let estPhoto = EstablishmentPhoto(context: container.viewContext)
        estPhoto.date = Date()
        estPhoto.photo = estPhotoItem.photoData
        if let estPhotoID = estPhotoItem.establishmentID {
            estPhoto.establishmentID = estPhotoID
        }
        estPhoto.uuid = estPhotoItem.uuid
        save()
    }

    func fetchReviews(by identifier: Int) -> [ReviewItem] {
        let moc = container.viewContext
        let request = Review.fetchRequest()
        let predicate = NSPredicate(format: "establishmentID = %i", identifier)
        var reviewItems: [ReviewItem] = []
        request.sortDescriptors = [NSSortDescriptor(key: "date", ascending: false)]
        request.predicate = predicate
        do {
            for review in try moc.fetch(request) {
                reviewItems.append(ReviewItem(review: review))
            }
            return reviewItems
        } catch {
            fatalError("Failed to fetch reviews: \(error)")
        }
    }

    func fetchRestPhotos(by identifier: Int) -> [EstablishmentPhotoItem] {
        let moc = container.viewContext
        let request = EstablishmentPhoto.fetchRequest()
        let predicate = NSPredicate(format: "establishmentID = %i", identifier)
        var estPhotoItems: [EstablishmentPhotoItem] = []
        request.sortDescriptors = [NSSortDescriptor(key: "date", ascending: false)]
        request.predicate = predicate
        do {
            for estPhoto in try moc.fetch(request) {
                estPhotoItems.append(EstablishmentPhotoItem(establishmentPhoto: estPhoto))
            }
            return estPhotoItems
        } catch {
            fatalError("Failed to fetch establishment photos: \(error)")
        }
    }


    private func save() {
        do {
            if container.viewContext.hasChanges {
                try container.viewContext.save()
            }
        } catch let error {
            print(error.localizedDescription)
        }
    }
}

extension CoreDataManager {
    static var shared = CoreDataManager()
}
```

This algorithm allows any user review or photo to be stored to the user's device, whether it's an iPad or an iPhone 7.0 or later. This allows for data sharing of any review or user experience content should the app be accepted by the Apple App Store specialists.

3. ExploreViewController:

```
import UIKit
import MapKit


// ExploreViewController is a UIViewController and a UICollectionViewDelegate,
// since it both provides for a ViewController of the various types of craft-beverages and
// it provides for a segue (a delegate) to the DrinkingEstablishmentCells
class ExploreViewController: UIViewController, UICollectionViewDelegate {

    @IBOutlet weak var collectionView: UICollectionView!
    let manager = ExploreDataManager()
    var selectedCity: LocationItem?
    var headerView: ExploreHeaderView!

    override func viewDidLoad() {
        super.viewDidLoad()
        initialize()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        navigationController?.setNavigationBarHidden(true, animated: false)
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        switch segue.identifier! {
        case Segue.locationList.rawValue:
            showLocationList(segue: segue)
        case Segue.establishmentList.rawValue:
            showEstablishmentList(segue: segue)
        default:
            print("Segue not added.")
        }
    }

    // Segue check
    override func shouldPerformSegue(withIdentifier identifier: String, sender: Any?) -> Bool {
        if identifier == Segue.establishmentList.rawValue,
           selectedCity == nil {
            showLocationRequiredAction()
            return false
        }
        return true
    }
}

// MARK: Private Extension
private extension ExploreViewController {
    func initialize() {
        manager.fetch()
        setupCollectionView()
    }

    // "Beer", "Spirits", "Wine" display properties
    func setupCollectionView() {
        let flow = UICollectionViewFlowLayout()
        flow.sectionInset = UIEdgeInsets(top: 7, left: 7, bottom: 7, right: 7)
        flow.minimumInteritemSpacing = 0
        flow.minimumLineSpacing = 7
        collectionView.collectionViewLayout = flow
    }

    func showLocationList(segue: UIStoryboardSegue) {
        guard let navController = segue.destination as? UINavigationController,
```

```
            let viewController = navController.topViewController as? LocationViewController
else {
                return
            }
        viewController.selectedCity = selectedCity
    }

    // Establishments cities in the Austin and 10 surrounding areas take the form, e.g.,
"Fredericksburg, TX"
    func showEstablishmentList(segue: UIStoryboardSegue) {
        if let viewController = segue.destination as? DrinkingEstablishmentsListViewController,
           let city = selectedCity,
           let index = collectionView.indexPathsForSelectedItems?.first?.row {
            viewController.selectedTags = manager.exploreItem(at: index).name
            viewController.selectedCity = city
        }
    }

    // Flag to ensure user selects a location first before selecting a craft-beverage type
    func showLocationRequiredAction() {
        let alertController = UIAlertController(title: "Location Needed", message: "Please select
a location.", preferredStyle: .alert)
        let okAction = UIAlertAction(title: "Ok", style: .default, handler: nil)
        alertController.addAction(okAction)
        present(alertController, animated: true, completion: nil)
    }

    @IBAction func unwindLocationCancel(segue: UIStoryboardSegue) {
    }

    @IBAction func unwindLocationDone(segue: UIStoryboardSegue) {
        if let viewController = segue.source as? LocationViewController {
            selectedCity = viewController.selectedCity
            if let location = selectedCity {
                headerView.locationLabel.text = location.cityAndState
            }
        }
    }
}


// View Control Delegate (segue) layaout for respective Apple devices
extension ExploreViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
        var columns: CGFloat = 2
        if Device.isPad || (traitCollection.horizontalSizeClass != .compact) {
            columns = 2
        }
        if Device.isPhone || (traitCollection.horizontalSizeClass != .compact) {
            columns = 1
        }
        let viewWidth = collectionView.frame.size.width
        let inset = 7.0
        let contentWidth = viewWidth - inset * (columns + 1)
        let cellWidth = contentWidth / columns
        let cellHeight = cellWidth
        return CGSize(width: cellWidth, height: cellHeight)
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, referenceSizeForHeaderInSection secton: Int) -> CGSize {
        return CGSize(width: collectionView.frame.width, height: 100)
    }
}

// MARK: UICollectionViewDataSource
extension ExploreViewController: UICollectionViewDataSource {
    func collectionView(_ collectionView: UICollectionView, viewForSupplementaryElementOfKind
kind: String, at indexPath: IndexPath) -> UICollectionReusableView {
        let header = collectionView.dequeueReusableSupplementaryView(ofKind: kind,
withReuseIdentifier: "header", for: indexPath)
```

```
        headerView = header as? ExploreHeaderView
        return headerView
    }

    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int)
-> Int {
        manager.numberOfExploreItems()  // Number of app panes
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath)
-> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "exploreCell", for:
indexPath) as! ExploreCell
        let exploreItem = manager.exploreItem(at: indexPath.row)
        cell.exploreImageView.image = UIImage(named: exploreItem.image!)
        return cell
    }
}
```

This class is a collection of algorithms and method calls from within and without the class itself which together allow for the EXPLORE PATH to be rendered. The EXPLORE PATH, detailed in the "Acceptance Test Cases.pdf" and in the **AustinDrinks** State Chart diagram, is what *is the opening screen of* **AustinDrinks**. The "AustinDrinks Full Demo.mov" also provides a full walk through of all the primary "pages" the app Start Chart path-lines lead through. Each of these items may be found here: https://github.com/Austin-Faulkner/AustinDrinks_iOSApp.git

4. LocationViewController:

```
import UIKit

// PRovides for the menu of 11 cities in the Austin, TX and surrounding areas in the Hill Country
class LocationViewController: UIViewController {

    let manager = LocationDataManager()
    var selectedCity: LocationItem?

    @IBOutlet weak var tableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()
        initialize()
    }

    // Sets a Checkmark next to the chosen city such there there exist a 1-1 mapping between user
choice
    // and craft-beverage city options
    func setCheckmark(for cell: UITableViewCell, location: LocationItem) {
        if selectedCity == location {
            cell.accessoryType = .checkmark
        } else {
            cell.accessoryType = .none
        }
    }
}

// MARK: Private Extension
private extension LocationViewController {
    func initialize() {
        manager.fetch()
        title = "Choose a destination"
        navigationController?.navigationBar.prefersLargeTitles = true
    }
}

// MARK: UITableViewDataSource
extension LocationViewController: UITableViewDataSource {

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        manager.numberOfLocationsItems()
```

```
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "locationCell", for: indexPath)
        let location = manager.locationItem(at: indexPath.row)
        cell.textLabel?.text = location.cityAndState
        setCheckmark(for: cell, location: location)
        return cell
    }

    // ADDED:
    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> Int {
        let cell = tableView.dequeueReusableCell(withIdentifier: "locationCell", for: indexPath)
        let location = manager.locationItem(at: indexPath.row)
        setCheckmark(for: cell, location: location)
        return Int((location.city)!)!
    }
}

// MARK: UITableViewDelegate
extension LocationViewController: UITableViewDelegate {

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        if let cell = tableView.cellForRow(at: indexPath) {
            cell.accessoryType = .checkmark
            selectedCity = manager.locationItem(at: indexPath.row)
            tableView.reloadData()
        }
    }
}
```

This class is a set of class attributes and methods that allow for one to choose a user-selected destination such as Austin, TX or, say, Fredericksburg, TX. The algorithm is simple and poses a 1-1 map between user-selection and app-destination. Thus, *there can only be 1, and only 1, destination so the function is well-defined when it comes to displaying any set of pages/views or application* PATHS *pertaining to* ***where*** *the breweries, distilleries, or wineries are located.* For the sake of simplicity one is not allowed to choose more than one destination, as is indicated by the setCheckmark() function. At least one, and only one, check mark must be placed by a user's choice of destination.

5. DrinkingEstablishmentListViewController:

```
import UIKit

class DrinkingEstablishmentsListViewController: UIViewController, UICollectionViewDelegate {

    private let manager = EstablishmentDataManager()

    var selectedEstablishment: EstablishmentItem? // Allows for establishment selection on
Drinking Establishments
                                                  // List View page such that a delegate (a
segue) can allow for a
                                                  // transition from the Establishment List View
to the Establishment Detail View.
    var selectedCity: LocationItem? // City selected from Explore PATH
    var selectedTags: String? // Type of craft-beverage selected on Explore Path

    @IBOutlet var collectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if let identifier = segue.identifier {
            switch identifier {
            case Segue.showDetail.rawValue:
                showEstablishmentDetail(segue: segue)
            default:
```

```
                print("Segue not added.")
            }
        }
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        initialize()
    }

    // TODO: moded feature of the List View for establishments; 'GO' moved to Establishment
Detail View, and is renamed 'Website'.
    @IBAction func goButton(_ sender: UIButton) {
        UIApplication.shared.open(URL(string: "https://www.hackerrank.com/")! as URL, options:
[:], completionHandler: nil)
    }
}

// MARK: Private Extension
private extension DrinkingEstablishmentsListViewController {
    func initialize() {
        createData()
        setupTitle()
        setupCollectionView()
    }

    // List View Flow properties
    func setupCollectionView() {
        let flow = UICollectionViewFlowLayout()
        flow.sectionInset = UIEdgeInsets(top: 7, left: 7, bottom: 7, right: 7)
        flow.minimumInteritemSpacing = 0
        flow.minimumLineSpacing = 7
        collectionView.collectionViewLayout = flow
    }

    func showEstablishmentDetail(segue: UIStoryboardSegue) {
        if let viewController = segue.destination as? EstablishmentDetailViewController,
           let indexPath = collectionView.indexPathsForSelectedItems?.first {
            selectedEstablishment = manager.establishmentItem(at: indexPath.row)
            viewController.selectedEstablishment = selectedEstablishment
        }
    }

    func createData() {
        guard let city = selectedCity?.city,
              let tag = selectedTags else {
                  return
              }
        manager.fetch(location: city, selectedEthanol: tag) {
            establishmentItems in
            if !establishmentItems.isEmpty {
                collectionView.backgroundView = nil
            } else {
                let view = NoDataView(frame: CGRect(x: 0, y: 0, width:
collectionView.frame.width, height: collectionView.frame.height))
                view.set(title: "Drinking Establishments", desc: "No establishments found.",
accomm: "No accommodations found.", rating: "0.0", count: "0")
                collectionView.backgroundView = view
            }
            collectionView.reloadData()
        }
    }

    func setupTitle() {
        navigationController?.setNavigationBarHidden(false, animated: false)
        title = selectedCity?.cityAndState
        navigationController?.navigationBar.prefersLargeTitles = true
    }
}

extension DrinkingEstablishmentsListViewController: UICollectionViewDelegateFlowLayout {
```

```
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
        var columns: CGFloat = 1
        if Device.isPad {
            columns = 2
        } else {
            columns = traitCollection.horizontalSizeClass == .compact ? 1 : 2
        }
        let viewWidth = collectionView.frame.size.width
        let inset = 9.0
        let contentWidth = viewWidth - inset * (columns + 1)
        let cellWidth = contentWidth / columns
        let cellHeigth = 312.0
        return CGSize(width: cellWidth, height: cellHeigth)
    }
}

// MARK:: UIColletionViewDataSource
extension DrinkingEstablishmentsListViewController: UICollectionViewDataSource {
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int)
-> Int {
        manager.numberOfEstablishmentItems()
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath)
-> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "establishmentCell",
for: indexPath) as! DrinkingEstablishmentCell
        let establishmentItem = manager.establishmentItem(at: indexPath.row)
        cell.titleLabel.text = establishmentItem.name
        if let tag = establishmentItem.subtitle {
            cell.tagLabel.text = tag
        }

        if let accommodation = establishmentItem.accommodation_attributes {
            cell.accommodationsLabel.text = accommodation
        }
        cell.ratingLabel.text = establishmentItem.rating_value
        cell.countLabel.text = establishmentItem.rating_count
        return cell
    }
}
```

This class and its algorithms provide for the ESTABLISHMENT LIST view (or page) in the EXPLORE PATH. This class renders the control functionality for finding a craft-beverage establishment in the city chosen in the initial part of the EXPLORE PATH by use of the +LOCATION function (See above).

6. MapViewController:

```
import UIKit
import MapKit

class MapViewController: UIViewController {

    @IBOutlet var mapView: MKMapView!
    private let manager = MapDataManager()
    var selectedEstablishment: EstablishmentItem?
    var selectedCity: LocationItem?

    override func viewDidLoad() {
        super.viewDidLoad()
        initialize()
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        switch segue.identifier! {
        case Segue.showDetail.rawValue: showEstablishmentDetail(segue: segue)
        default: print("Segue not added")
        }
    }
```

```
}

// MARK: Private Extension
private extension MapViewController {
    func initialize() {
        mapView.delegate = self
        manager.fetch { (annotations) in setupMap(annotations) }
    }

    func setupMap(_ annotations: [EstablishmentItem]) {
        mapView.setRegion(manager.currentRegion(latDelta: 0.5, longDelta: 0.5), animated: true)
        mapView.addAnnotations(manager.annotations)
    }

    func showEstablishmentDetail(segue: UIStoryboardSegue) {
        if let viewController = segue.destination as? EstablishmentDetailViewController,
           let establishment = selectedEstablishment {
            viewController.selectedEstablishment = establishment
        }
    }
}

// MARK: MKMapViewDelegate
extension MapViewController: MKMapViewDelegate {

    func mapView(_ mapView: MKMapView, annotationView view: MKAnnotationView,
calloutAccessoryControlTapped control: UIControl) {
        guard let annotation = mapView.selectedAnnotations.first else {
            return
        }
        selectedEstablishment = annotation as? EstablishmentItem
        self.performSegue(withIdentifier: Segue.showDetail.rawValue, sender: self)
    }

    func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {
        let identifier = "custompin"
        guard !annotation.isKind(of: MKUserLocation.self) else {
            return nil
        }

        let annotationView: MKAnnotationView
        if let customAnnotationView = mapView.dequeueReusableAnnotationView(withIdentifier:
identifier) {
            annotationView = customAnnotationView
            annotationView.annotation = annotation
        } else {
            let av = MKAnnotationView(annotation: annotation, reuseIdentifier: identifier)
            av.rightCalloutAccessoryView = UIButton(type: .detailDisclosure)
            annotationView = av
        }
        annotationView.canShowCallout = true
        if let image = UIImage(named: "custom-annotation") {
            annotationView.image = image
            annotationView.centerOffset = CGPoint(x: -image.size.width / 2, y: -image.size.height
/ 2)
        }
        return annotationView
    }
}
```

The MapViewController establishes the MAP PATH's properties and semantics. See the State Chart diagram here: https://github.com/Austin-Faulkner/AustinDrinks_iOSApp.git. Namely, the methods within the class together define how the map looks; where the pins are to be situated on the map according to GPS coordinates given in the Austin, TX JSON file, "Austin.json"; and how *one is able to click on the pin to then be able to be transferred to the DrinkingEstablishmentDetailView of the chosen pin*.

Taken together, these algorithms and UML-structural relationships define the engine of what **AustinDrinks** accomplishes.