**Instacart App System Design:**

**Understand the problem:**

* **MVP or final product –** Final Product
* **User base/category**: Anyone web or mobile users as consumer, not as merchant.
* **number of users & expected growth** – Let’s say 10 million active users so Exponential backoff and API rate limiting
* **Design wireframes** – Ignore for now
* **Team Size** – Let’s say infinite resources
* **Client vs Client+ API vs Client+API+Backend :** Client + API + BE
* **Device and OS Support, Cross Platform** – Let’s say all platforms, just phones for now.
* **Which countries, emerging markets?** – Let’s say yes. So, we need high caching, limit the number and frequency of user requests, control the app size
* **Maximum orders during business days, not after store close.**

**Define the scope:**

|  |  |  |
| --- | --- | --- |
| **Functional** | **Non-Functional** | **Out of Scope/Assumptions** |
| Show list of stores based on my location, search for a store   * Only read existing catalog/partners * Adding product/category/partner not in scope. * Add Pagination | Security:   * PII data protection using keychain * Secure BE communication using HTTPS, TLS, Cert pinning * Use .xcconfig file to store API Keys and ignore using gitigore, or use private cloudKit DB | Monitoring   * Analytics, Crash reporting and logging * Firebase Analytics, Lighthouse, Firebase Crashlytics, CocoaLumberJack   Problems with Firebase: No support in China so better to create tools inhouse. |
| View all categories in a store + details  View all products in any category + details  View Catalog of a store -> shows top categories with top product | Availability:   * Offline mode: Based on the requirement of the app, offline mode can’t be supported. But already loaded feed can still be browsed. Cart management from different device isn’t possible with offline mode. | Deployment   * CI/CD: Jenkins + Fastlane, travisCI, CircleCI * Experiementation: ELMO: Firebase A/B Testing, Optimizely, |
| Add/remove product to cart  Change quantity, checkout | Scalability:   * Modularize the app in Networking, Interfaces, UI Interfaces, Data Interfaces, Common utilities etc. | Login and Authentication,  Delivery + Return service,  suggestions based on ML,  Inventory/merchant management,  Payment system, gateway,  Store the FI for future purchase,  User Profile Management, |
|  | Performance:   * Infinite scroll handling using pagination * Prefetch images + caching * CDN for images and static content |  |
|  | Accessibility + L10n + I18n   * Proper contrast, 44 pt. target, Dynamic font size, voice over * LTR languages, and translations * Verbose languages like DE |  |
| Order tracking | Testing:   * Dependency Injection * Regression tests at the app level to ease release process |  |
|  |  |  |
|  |  |  |

**Identify technical requirements**

* Client architecture: MVVM Architecture (or Upgraded version of MVC)
  + Pros:
  + Its more reactive, ViewModel changes based on Model, and binds the action to views. On state change, view notifies ViewModel and that updates Model if necessary
  + the view layer and model layer are completely decoupled, so easy to work in parallel for the team
  + Easy testing using dependency injection
  + No UI in ViewModel, so no Import of UIKit, and can be used across iOS, tvOS, and macOS
  + Cons:
  + Learning curve,
  + can be hard to trace and debug, because UI updates happen through binding instead of method calls.
  + Massive ViewModel if childViewModels aren’t created.
* Networking:
  + **REST VS GraphQL** API: if require multiple network calls and customization of BE structure, prefer GraphQL, gives you ability to define how you want the response, and no extra things come on client side.
  + **HTTP VS Sockets**: No live updates, or low latency requirement, Unidirectional communication, HTTP is okay. Else use polling, websockets, heartbeats etc. like for chat apps.
  + **Slow Networks**: No special handling required.
  + **Background tasks**: No need of background downloading.
  + **Reachability/Offline mode**: AFNetworking or [Reachability](https://github.com/ashleymills/Reachability.swift). No Offline mode.
* Communication: **Protocols**, closures, Notifications, Publishers (PromiseKit, Combine etc if you support iOS 13 and above only)
* Navigation: Ideally View Model handles the navigation, using common Navigator class, which conforms to **Navigation protocols**. Deep Link + Universal Link support if needed.
* Storage: No Local storage needed; offline features are not available so.
* Third Party Dependencies: sd\_webImage, RX\_Swift/PromiseKit, SnapKit, Alamofire, Adjust, SwiftLint, Firebase, etc. ( I personally avoid using those )

**Propose a high-level design**

* Data Entities: User, Store, Item, Category, Cart, Order, StoreDetail, CategoryDetail, ItemDetail,
* BE Architecture: Micro service architecture as mentioned in eCommerce-BE.drawio file.
* Client Architecture: Micro service architecture as mentioned in eCommerce-Client-HLD.drawio file.

**Services:**

**StoreService:** Reverse Index of the location (location -> store mapping has to be stores somewhere, like 95134 -> [Walmart, safeway, Krogger])

/showNearbyStores(location, radius) -> [StoreID]

/searchStore(text, location, radius) -> StoreID?

/storeDetails(storeID) -> StoreDetails -> Name, Address, Fastest Delivery By, etc.

**CatalogService**:

/getCatalogView(storeID) -> shows top categories with top product

/getCategories(storeID) -> [CategoryID]

/getItems(storeID, categoryID) -> [ItemID]

/categoryDetails(id) -> name, image

/itemDetails(id) -> name, price, weight, image

/getQuantity(itemID) -> Int?

/search(text, storeID?) -> only search based on product -> [ItemId]

**CartService**:

cartId <-> storeId

/add(cartId, ItemId, qty? = 1)

/remove(cartId, itemId)

/update(cartId, itemId, qty)

/getTotal(cartId)

/checkout(cartId)

NotificationService: OOScope

PaymentService: OOScope

OrderTracking: OOScope

FulfillmentService: OOScope

UserService: OOScope

**Deep dive into one component**

**Wrap up**