How Using NNNetwork simplified i18n config

Objectives

- Configure iOS app for IT/ES (help urls, account creation form, country config, etc.)
- Create a new format for service and web urls: `es.nextdoor.com` and `es-api.nextdoor.com` while still maintaining legacy urls for other countries that support the tld format (e.g. `api.nextdoor.de`)
  - Can’t always purchase tld url (e.g fr.nextdoor.com). This can lead to inconsistent and confusing url formats that require special casing
  - Purchasing tlds requires configuration of certificates. Proliferation of hosted zones and certificates.
  - With subdomains, you can have up to 20 subdomains for a given domain and don’t have to configure certs. All DNS entries present under one domain and hosted zone for nextdoor.com. Can have more consistent formatting across services and countries.
  - Plan to backport legacy urls to subdomain
  - Systems team will come up with strategy for growing past 20 subdomains
- Have a lightweight system for performing GETs and POSTs and modifying headers in existing requests.
  - Speeding up autocomplete. Need to modify request to `mobile/v1/addresses` endpoint

Problems with i18n config in iOS

- Messy functions for constructing service endpoints; very brittle. Relying on Strings for url construction; can be risky and error-prone
- Network calls very brittle and difficult to debug. Implement several protocols, most of which are deprecated. Have to trace down several files.
- Scattered and duplicate enums for defining country specific features in iOS repo. Have to be very careful when adding a new country. One feature might break and difficult to find out during development cycle.

Brittle, hard to debug functions

```
(void)configureAPIServerBaseURLForEndpoint:(NDAPIServerEndPointType)endPoint
                        useHardcodedHost:(BOOL)useHardcodedHost
                              serverType:(NDAPIServerType)serverType {
  static NSString *NDAPIServerScheme = @"https://";
  static NSString *NDAPIServerPath = @"";
  static NSString *NDAPIServerSubDomain = @"";
  static NSString *NDAPIServerPort = @"";
  static NSString *NDAPIVersion = @"";
  static dispatch_once_t *predicate;

  NSString *devPort = @"";
  switch (serverType) {
      case NDAPINextdoorServer:
          NDAPIServerPath = @"mobile";
          NDAPIServerSubDomain = @"api";
          NDAPIVersion = @"v1";
          devPort = @":8000";
          predicate = &apiServerEndPointOnceToken;
          break;
      case NDAPIFlaskServer:
          NDAPIServerPath = @"";
          NDAPIServerSubDomain = @"tracking";
          NDAPIVersion = @"";
          devPort = @":8300";
          predicate = &flaskApiServerEndPointOnceToken;
          break;
      case WebServer:
          NDAPIServerPath = @"";
          NDAPIServerSubDomain = @"";
          NDAPIVersion = @"";
          devPort = @":8000";
          predicate = &webServerEndPointOnceToken;
          break;
      case NDAPIRingbearerServer:
          NDAPIServerPath = @"";
          NDAPIServerSubDomain = @"auth";
          NDAPIVersion = @"v1";
          devPort = @":8133";
          predicate = &ringbearerServerEndPointOnceToken;
          break;
      case NDAPIRegionServer:
          NDAPIServerPath = @"";
          NDAPIServerSubDomain = @"region";
          NDAPIVersion = @"v1";
          devPort = @":8000";
          predicate = &regionServerEndPointOnceToken;
          break;
      case NDAPISocketServer:
          NDAPIServerPath = @"";
          NDAPIServerSubDomain = @"sockets";
          NDAPIVersion = @"";
          devPort = @":8077";
          predicate = &socketServerEndPointOnceToken;
          break;
  }

  // In a preview environment context, all network access must be through
  // URL https://<preview-name>.nextdoor-test.com/
  // The subdomains such as 'auth', 'region', 'sockets' etc are not available.
  if (endPoint == NDAPIServerEndPointTypeDynamic) {
      NDAPIServerSubDomain = @"";
      useHardcodedHost = true;
  }

  dispatch_once(predicate, ^{
      NSString *subDomain = NDAPIServerSubDomain;
      NSString *port = NDAPIServerPort;
      NSString *host;
      if (useHardcodedHost) {
          host = [self NextdoorHardcodedHostForEndPoint:endPoint];
      } else {
          host = [NDNetwork NextdoorHost];
          if (!host) {
              // Fall back to defaults if no host exists.
              host = [self NextdoorHardcodedHostForEndPoint:endPoint];
          }
      }
      NSString *scheme = NDAPIServerScheme;
      switch (endPoint) {
          case NDAPIServerEndPointTypeDev:
              port = devPort;
              scheme = @"http://";
              if (serverType == NDAPIRingbearerServer) {
                  scheme = @"https://";
              }
              break;
          case NDAPIServerEndPointTypeNone:
          case NDAPIServerEndPointTypeStaging:
          case NDAPIServerEndPointTypeProduction:
          case NDAPIServerEndPointTypeDynamic:
              break;
      }

      NSString *absoluteURLString =
          [NSString stringWithFormat:@"%@%@%@%@%@%@%@%@/",
              scheme,
              subDomain,
              [subDomain hasValue] ? @"." : @"",
              host,
              port,
              [NDAPIServerPath hasValue] ? @"/" : @"",
              NDAPIServerPath,
              [NDAPIVersion hasValue] ? @"/" : @"",
              NDAPIVersion];

      NSURL *baseURL = [NSURL URLWithString:absoluteURLString];
      switch (serverType) {
          case NDAPINextdoorServer:
              NDNextdoorAPIServerBaseURL = baseURL;
              break;
          case NDAPIFlaskServer:
              NDFlaskAPIServerBaseURL = baseURL;
              break;
          case WebServer:
              WebServerBaseURL = baseURL;
              break;
          case NDAPIRingbearerServer:
              NDRingbearerServerBaseURL = baseURL;
              break;
          case NDAPIRegionServer:
              RegionServerBaseURL = baseURL;
              break;
          case NDAPISocketServer:
              SocketServerBaseURL = baseURL;
              break;
      }
  });
}
```

Better

```swift
public enum Country: String {
    case us, nl, gb, fr, de, it, es

    public init?(tld: String) {
        switch tld {
        case "com":
            self = .us
        case "co.uk":
            self = .gb
        default:
            if let cty = Country(rawValue: tld) {
                self = cty
            } else {
                return nil
            }
        }
    }
```

```swift
func apiUrl(scheme: Scheme, name: String) -> String {
    switch self {
    case .dev(let country):
        switch country {
        case .us:
            return "\(Scheme.http.rawValue)api.\(Host.dev(country: country).host):8000"
        default:
            return "\(Scheme.http.rawValue)\(country.rawValue)-api.\(Host.dev(country: country).host):8000"
        }
    case .preview(let country):
        switch country {
        case .us:
            return "\(scheme.rawValue)\(name).\(Host.preview(country: country).host)"
        default:
            return "\(scheme.rawValue)\(country)-\(name).\(Host.preview(country: country).host)"
        }
    case .staging(let country):
        switch country {
        case .us, .nl, .gb, .de, .fr:
            return "\(scheme.rawValue)api.\(Host.staging(country: country).host)"
        default:
            return "\(scheme.rawValue)\(country.rawValue)-api.\(Host.staging(country: country).host)"
        }
    case .production(let country):
        switch country {
        case .us, .nl, .gb, .de, .fr:
            return "\(scheme.rawValue)api.\(Host.production(country: country).host)"
        default:
            return "\(scheme.rawValue)\(country.rawValue)-api.\(Host.production(country: country).host)"
        }
    }
}
```

Simple and clear function calls for iOS

```
func testDevIT() {
    [Environment.dev(country: .it), Environment(host: "it.localhost.com")!].forEach { (dev) in
        XCTAssertEqual(Service.api.url(for: dev).absoluteString, "http://it-api.localhost.com:8000")
        XCTAssertEqual(Service.auth.url(for: dev).absoluteString, "https://auth.localhost.com:8133")
        XCTAssertEqual(Service.region.url(for: dev).absoluteString, "https://region.localhost.com:8133")
        XCTAssertEqual(Service.socket.url(for: dev).absoluteString, "http://it-sockets.localhost.com:8177")
        XCTAssertEqual(Service.web.url(for: dev).absoluteString, "http://it.localhost.com:8000")
    }
}

func testDevES() {
    [Environment.dev(country: .es), Environment(host: "es.localhost.com")!].forEach { (dev) in
        XCTAssertEqual(Service.api.url(for: dev).absoluteString, "http://es-api.localhost.com:8000")
        XCTAssertEqual(Service.auth.url(for: dev).absoluteString, "https://auth.localhost.com:8133")
        XCTAssertEqual(Service.region.url(for: dev).absoluteString, "https://region.localhost.com:8133")
        XCTAssertEqual(Service.socket.url(for: dev).absoluteString, "http://es-sockets.localhost.com:8177")
        XCTAssertEqual(Service.web.url(for: dev).absoluteString, "http://es.localhost.com:8000")
    }
}
```

Old Network Calls

```
// MARK: - Helpers
private func fetchAddresses(postalCode: String) {
    let addressesRequest = AddressSuggestionsRequest(postalCode: postalCode)
    self.network.send(
        addressesRequest,
        successHandler: {
            (responseObject:DeprecatedNetworkResponseProtocol!) in
            guard let addressResponse = responseObject as? AddressSuggestionsResponse,
                let addressSearchResults = addressResponse.addressResults else {
                    return
            }

            self.searchResults = addressSearchResults
            self.filteredSearchResults = addressSearchResults
            // We must call notifySearchTextChanged at this point, because the request may
            // have completed AFTER the user has typed in some search text. In this case, we
            // must update the filtered results, and reload them.
            self.notifySearchTextChanged(searchText: self.searchText)
    }, failureHandler: {
        (_:DeprecatedNetworkResponseProtocol!, _: NetworkError!) in
        // Silently fail
    })
}
```

```
class AddressSuggestionsRequest: DeprecatedNetworkRequest, DeprecatedNetworkRequest

    let postalCode: String

    init(postalCode: String) {
        self.postalCode = postalCode
        super.init()
    }

    func deprecatedResponseObject(withJsonDict dict: [AnyHashable: Any]!) -> Deprec
        return AddressSuggestionsResponse(deprecatedJsonDict: dict)
    }

    var deprecatedMethod: String! {
        return "addresses"
    }

    override var deprecatedParamsDict: [AnyHashable: Any]! {
        return [
            "postal_code": self.postalCode,
            "hood_names": 1
        ]
    }
```

New Network Calls

```
// MARK: - Helpers
private func fetchAddresses(postalCode: String) {
    let params = FindResidenceParams(postalCode: postalCode)
    var request = Request<FindResidenceParams, NoResponse>(service: .api, path: "/mobile/v1/addresses",
                                                           params: params, serializer: .url)
    request.headers = ["Accept": "application/vnd.nextdoor/schema/suggested-address.v3+json"]
    request.shouldQueue = false
    nnnetwork.get(request) { result in
        switch result {
        case .Completion(let jsonResponse as [AnyHashable: Any]):
            let response = AddressSuggestionsResponse(deprecatedJsonDict: jsonResponse)
            var error: OldNetworkError?
            response?.deprecatedProcessAndPersistResponse(&error)
            let addressSearchResults = response!.addressResults
            self.searchResults = addressSearchResults!
            self.filteredSearchResults = addressSearchResults!
            self.notifySearchTextChanged(searchText: self.searchText)
        default: ()
        }
    }
}
```

Future Refactors

Configure iOS app for new country	i18n_infra	1. Add tld and locale to tldDict and intlHelpDeskLocaleDict in NDURLManager **(done 15 countries)** 2. Add country code to getLanguageURLDict in NDURLManager **DONE** 3. Add country name to ND.strings **(done for 15 countries)** 4. Add country code to country name mapping in NDStringFactory **(done for 15 countries)** 5. Add Account Creation Form details to NDAccountCreationFormView. **Done** 6. Add new domains to Nextdoor.entitlements, Nightly.entitlements, and RC.entitleme to enable deep linking. **Done** 7. Add language to the Nextdoor, NDUIKit, NDModel, and NDFoundation project setti under Info → Localizations. **Done** 8. Add entry to "locales" in mobile/etc/smartling_config.json - **NOTE**: this smartling config step needs to be done separately for Android. **DONE**

Benefits Gained By Leveraging NNNetwork

- One country enum that mobile repo can read from, and reduce code duplication and human error
- Easier to debug and modify endpoint construction, as networking code exists in one place now, is thoroughly tested, and bound to strict types
- Network calls much more lightweight, don’t have to trace down several layers of classes and deprecated network code. Can easily modify network requests and perform gets and posts easily.