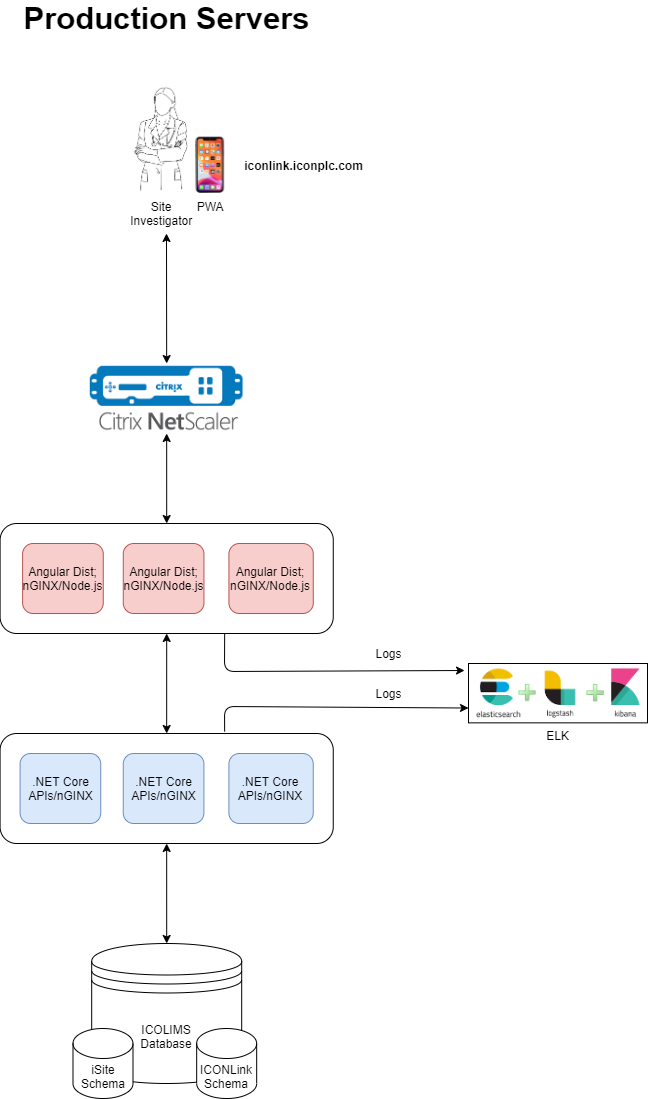
Document about Icon Links Project

1. Architecture

Progressive Web APP (PWA) is an app that's built using web platform technologies, but that provides a user experience like that of a platform-specific app.

Like a website, a PWA can run on multiple platforms and devices from a single codebase. Like a platform-specific app, it can be installed on the device, can operate while offline and in the background, and can integrate with the device and with other installed apps.



1.2. Angular frontend

.

.

└── app/

├── modules/

│ ├── account/

│ │ ├── login

│ │ └── request-account

│ └── home

├── shared/

│ ├── components/

│ │ ├── button

│ │ ├── dialog

│ │ └── input

│ ├── guard

│ ├── interceptors

│ ├── layouts/

│ │ ├── header

│ │ └── footer

│ ├── models/

│ │ └── auth

│ ├── services/

│ │ └── auth.service

│ └── utils/

│ ├── date

│ └── list

├── store/

│ └── auth/

│ ├── auth.actions

│ ├── auth.effects

│ ├── auth.reducer

│ ├── auth.selectors

│ └── auth.state

├── styles/

│ ├── \_global.scss

│ ├── \_variables.scss

│ └── index.scss

├── assets/

│ ├── icons

│ └── image

├── environments

├── app-routing.module.ts

├── app.module.ts

├── app.component.ts

├── app.component.html

└── app.component.scss

**Modules**

Organize components representing each page or item of the menu in the header of your project.

Each module represents a specific feature or business domain. Example:

* Account: This module handles user account-related features such as login and account registration.
* Subjects: This module manages subjects and their related functionality.

**Shared**

The Shared module has path /src/app/shared.

There are many components, directives & pipes, which we may like to share across various modules. All these components should go into the shared module.

The commonly required angular modules like ( CommonModule, FormsModule, etc) or third-party modules can be imported here and re-exported.

* Components
* Services
* Models
* Layouts
* Guard
* Interceptors
* Utils

Layouts: It contains layout components like header, footer, and main, which provide consistent structure across different pages.

Components: common UI components like buttons, dialogs, inputs,...

Services: Holds various shared services, such as authentication, visitor, and subject services,...

Guard: manage accessibility of a route, protect routes based on condition, when the use is authenticated or has specific permissions

Interceptors: manage HTTP request and response. Example: attach JWT token

Utils: contains utility functions, such as date handling and list operations.

Models: represent data structure that define the shape of data used within the application

**Store**: State management uses NgRx. It manages the application's state in a predictable and centralized.

* **State**

Refers to the centralized data store that holds the entire application's state.

* **Selectors**

Are functions used to extract specific pieces of state from the store.

* **Effects**

Are a way to handle side effects in your application, such as making HTTP calls, updating local storage, or interacting with external APIs.

* **Reducers**

Are pure functions responsible for handling state changes in the store.

* **Actions:**

Are simple objects that represent events in your application.

**Styles**

It contains CSS files and other style-related resources for the application

**Assets**

It contains all image files used in the project such as logo, icon, and other image

**Environments**

It holds environment-specific configuration files for different environments (e.g., production, development)

1.3. DotNET Core back-end

The overview of Back-end structure code folder will be shown on the figure 1 below.

.

.

├── ICONLinkAPI/

│ ├── Configurations/

│ │ ├── ConfigAuthentication.cs

│ │ ├── ConfigDI.cs

│ │ └── ConfigExceptionMiddlewares.cs

│ ├── Controllers/

│ │ ├── UserController.cs

│ │ └── ...

│ ├── Middlewares/

│ │ ├── GlobalErrorHandler

│ │ └── ...

│ ├── Properties/

│ │ └── LaunchSetting.json

│ ├── appsettings.json

│ └── Program.cs

├── ICONLinkAppplication/

│ ├── Interfaces/

│ │ ├── Services/

│ │ │ ├── IUserService.cs

│ │ │ └── ...

│ │ └── Repositories/

│ │ ├── IGenericRepository.cs

│ │ ├── IUnitOfWork.cs

│ │ ├── IUserRepository.cs

│ │ └── ...

│ └── Services/

│ ├── UserService.cs

│ └── ...

├── ICONLinkDomain/

│ ├── DTOs/

│ │ ├── User.cs

│ │ └── ...

│ ├── Entities/

│ │ ├── ISiteSchema/

│ │ │ ├── User.cs

│ │ │ └── ...

│ │ └── ICOLIMSSchema/

│ │ └── ...

│ ├── Exceptions/

│ │ └── CustomException.cs

│ ├── Mappings/

│ │ └── UserMapper.cs

│ └── DependencyInjectionExtension.cs

└── ICONLinkInfrastructure/

├── Persistences/

│ ├── Contexts/

│ │ ├── ICONLinkContext.cs

│ │ └── ...

│ └── Repositories/

│ ├── GenericRepository.cs

│ ├── UnitOfWork.cs

│ ├── UserRepository.cs

│ └── ...

└── DependencyInjectionExtension.cs

Figure 1: The skeleton of structure code for ICONLink back-end

ICONLink Back-end Architecture was designed based on the Clean Architecture which was introduced by Microsoft. The overview of Clean Architecture is shown on Figure 2.

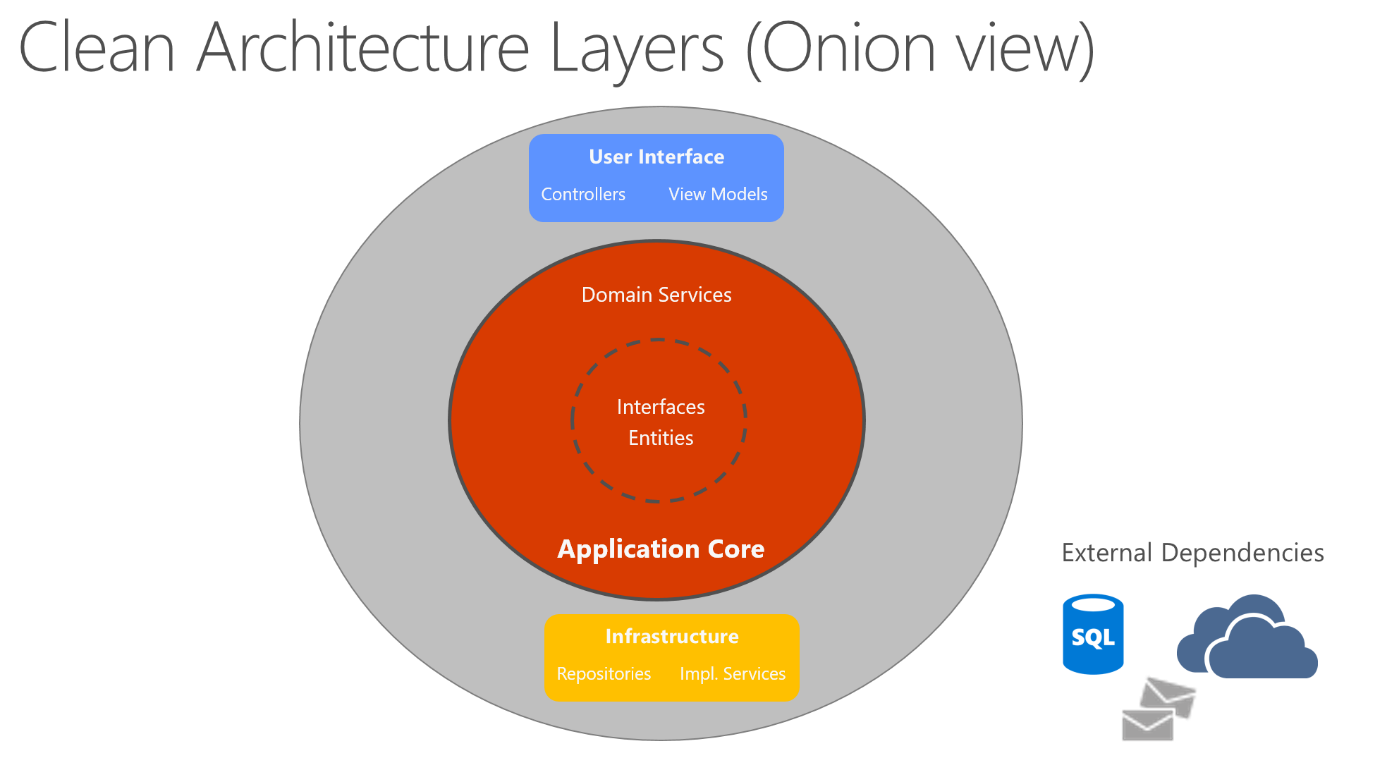


Figure 2: Clean Architecture Layers (Onion View) [1]

The ICONLink backend Architecture will have 4 main layers:

* **Domain Layer**: Contain the define of data structure of the application.
  + **DTOs folder**: Define the data structure of client’s request, and the API response that will be returned to the client.
  + **Entities folder**: Data structure of object that was mapped from Oracle database.
  + **Exceptions folder**: Data structure for some custom exceptions.
  + **Mappings folder**: Functionality for mapping or transferring the data between DTOs and Entities structure
* **Application Layer**: Contain business logic of the application
  + **Interfaces folder:** Interfaces of Repository, Service,...
  + **{Implemented}** **folder** (ex: Services Folder): Contain all of the implementations for all interfaces.
* **Infrastructure Layer**: Contain the **implementation of repositories**, the way to interact with Oracle database through **DbContext** (EF Core)
* **Presentation Layer**: Contain API application for setup of the endpoints to allow clients to retrieve the information and data from the server
  + **Controllers**: Routing the endpoints to specific action and execute the service.
  + **Configurations**: Some configurations for API application can work properly.
  + **Middlewares**: Some custom middlewares that will implement in API Application.

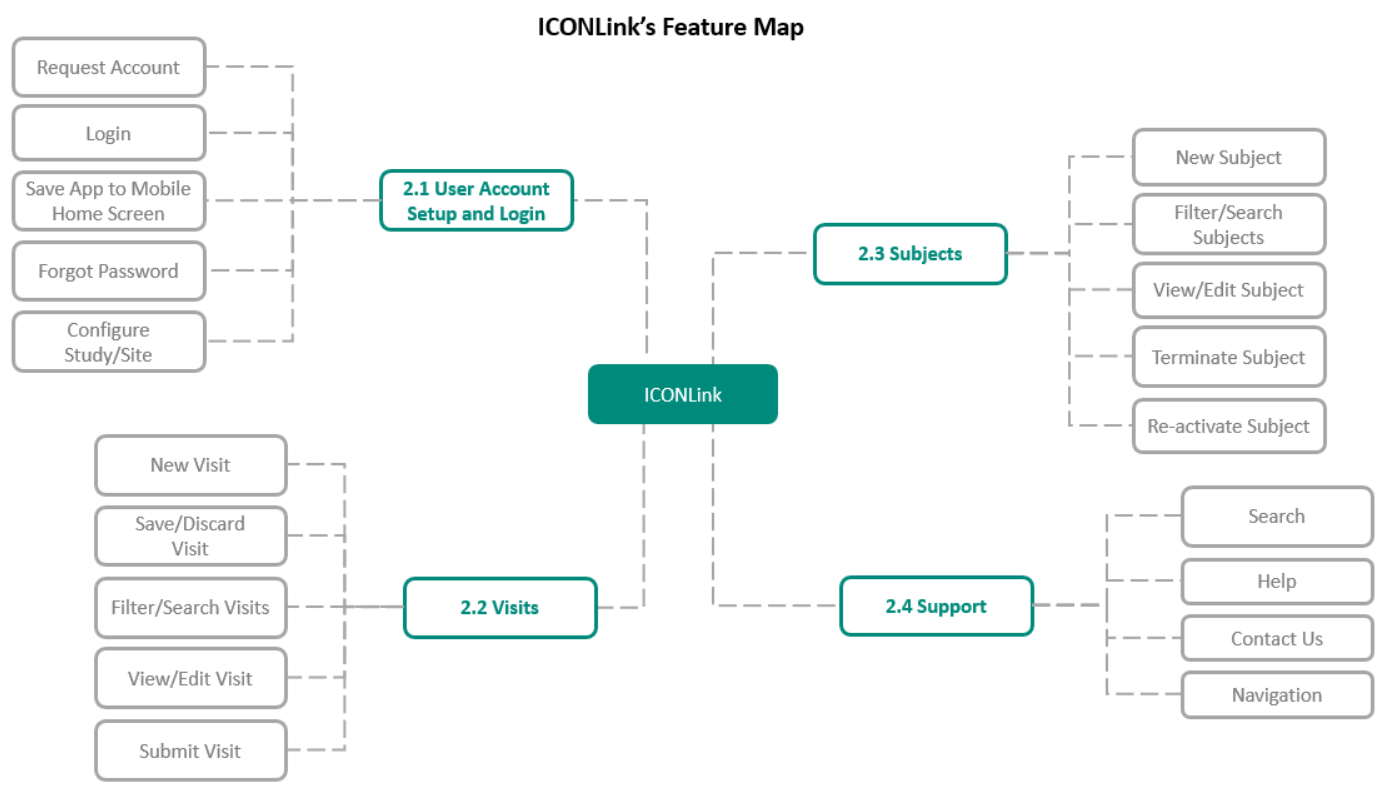
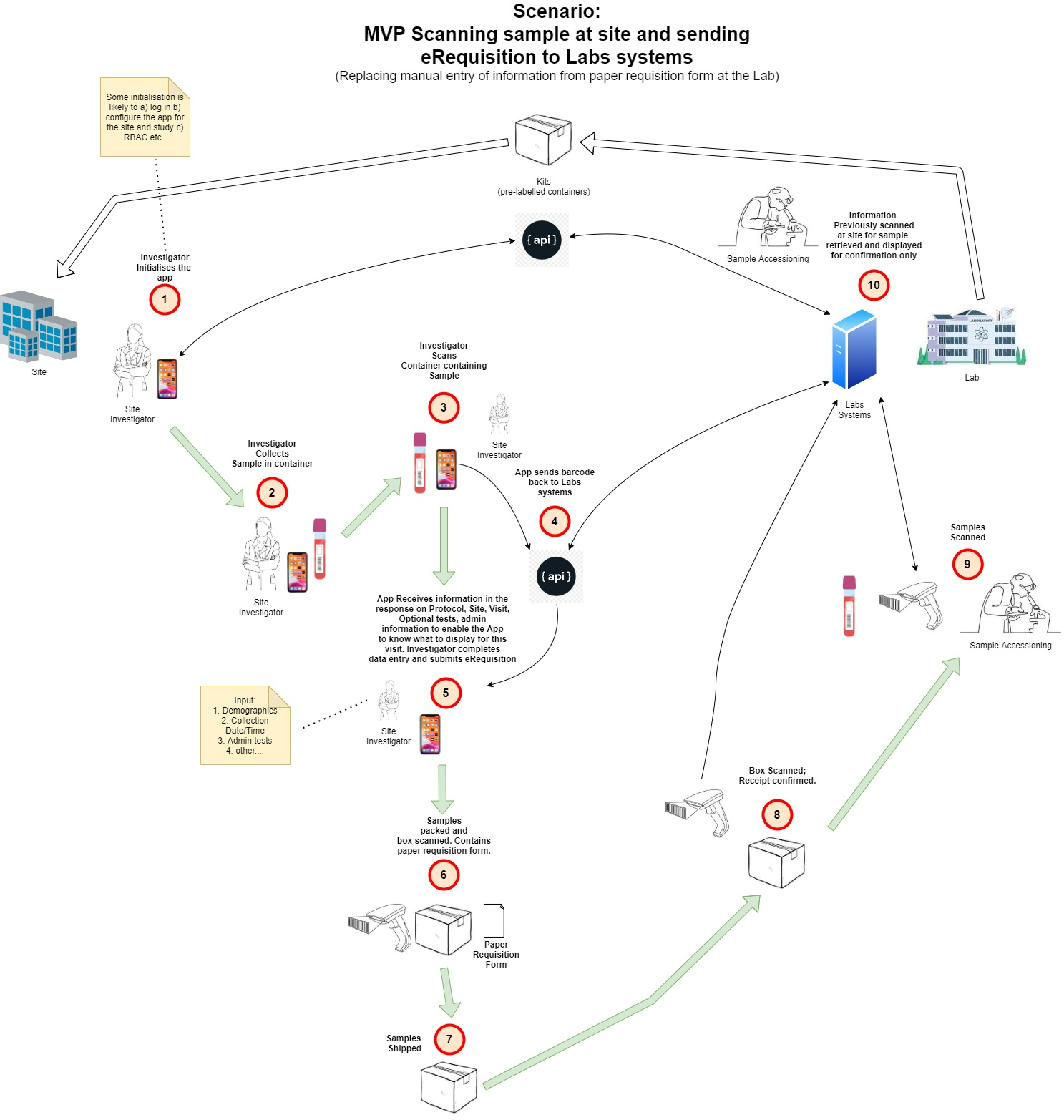
1.4. Oracle database

1.5. nGINX to loadbalance

1.6. Netscaler to dynamic slace & loadbalance in DC

1.7 Authentication using JWT

2. Features



3. Enviroments

4. Todo

5. References

[1]: Common web application architectures, https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures