### [BeTagged.Clients](https://github.com/cawstudios/BeTagged.Clients)

It is a monorepo consisting of the web and mobile client written in ReactJS and ReactNative.

**Nature of Repo:** Monorepo based on Lerna

**Framework:** ReactJS for Web Client and ReactNative for Mobile Client

**Node Version: 14.17.3  
React Native: 0.66.4  
React: 17.0.2**

**Files:**Mobile: 333  
Shared: 278  
Web: 219

**LOC:**

Mobile: 24127

Shared: 16770

Web: 23225

**Key parts of the codebase:**

| 1 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/docs> | The docs contain the coding standards and getting starting instructions for web client |
| --- | --- | --- |
| 2 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/web> | Contains the Web Client Codebase. |
| 3 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/mobile> | Contains the Mobile Client Codebase |
| 4 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/shared> | Contains the common code shared between web and mobile client |
| 5 |  | These four files are environment files for mobile. |
| 6 |  | These two files are environment files for web |
| 7 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/mobile/App> | It contains the ReactNative code for the mobile App.  The files have been organized in folders based on the functions. Eg, All components can be found in the “Components” folder.  The actual screens are organized under the “Screens” folder. Note, all the code related to connecting APIs will be found in the Shared package as it is common code between web and mobile. |
| 8 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/web/src> | It contains the ReactJS code for the web App.  The web client is divided into three key parent folders   1. App - This is where all the RJS startup code and routing logic lives. The components that build the website are also part of this folder. 2. Account-module - All the components and logic related to sign, sign in, and forgot password live in this folder. 3. Dashboard-module - Everything that powers the influencer and branch dashboard can be found under this folder   The files inside these three parent folders have been organized in sub-folders based on the functions. Eg: All components can be found in the “Components” folder.  Note, all the code related to connecting APIs will be found in the Shared package as it is common code between web and mobile. |
| 9 | <https://github.com/cawstudios/BeTagged.Clients/actions> | The GitHub actions for building the web can be found here. |
| 10 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/shared/src/store> | The Redux store configuration |
| 11 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/shared/src/assets/icons> | All the assets shared between web and mobile |
| 12 | <https://github.com/cawstudios/BeTagged.Clients/blob/develop/packages/shared/src/strings/LocalisedStrings.ts> | Note, BeTagged codebase doesnt hard code strings in the codebase directly. All strings are maintained in this file. |
| 13 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/shared/src/services/Logging> | All code related to logging. |
| 14 | <https://github.com/cawstudios/BeTagged.Clients/tree/develop/packages/mobile/App/components> | All the components that build up mobile app are found in this folder |

### 

### [BeTagged.CoreAPIs](https://github.com/cawstudios/BeTagged.CoreAPIs)

It is the main backend codebase written in C# .NET. It is based on the DotNET Core using the WebAPI Framework.

**Framework:** DotNet Core WebAPIs

**Version: 6.0**

**Files: 808**

**LOC: 261133**

**Overview:**

The project is an advanced ASP.NET Core web application designed for effective request handling and data processing. It integrates MediatR for CQRS, EF Core, and Dapper for data access, and employs the specification pattern for querying. Additionally, the project utilizes Hangfire with Redis for background task processing and managing recurring jobs, thereby enhancing performance and scalability. The code is containerized and deployed to AWS fargate as docker container.

The codebase follows the **MVC pattern**.

At the root, <https://github.com/cawstudios/BeTagged.CoreAPIs>, you will find key folders to organize the code

1. Common - All the common code, including utils, logger, DI configuration, can be found here. Note that no business logic or DB-related logic exists in this folder.
2. Data - This project has all code related to DB. We use Entity Framework Core 6 and dapper to connect to the DB. This project doesn’t contains any business logic.
3. DbSeeder - This is a command line tool to bootstrap the database with static data like countries, phone number codes etc.
4. Core - It has all the business logic.
5. Web - Has all the controllers for the APIs. This project only accept the request and delegate it to the Core project where actual business logic gets executed.

**Environments**:

1. Local - The environment to be used locally.
2. Development
3. Production

The enviroment is specified by **ASPNETCORE\_ENVIRONMENT** environment variable.

**Key Components:**

1. ASP.NET Core: Provides the framework for building the web application, ensuring high performance and compatibility with modern web standards.
2. MediatR for CQRS: Implements the CQRS pattern, separating read and write operations, improving scalability, and simplifying maintenance.
3. Entity Framework Core: Serves as the primary ORM tool for standard database operations, leveraging LINQ for queries and ensuring type safety.
4. Dapper: Used for complex queries where raw SQL execution and efficient data mapping to POCOs are necessary.
5. Specification Pattern with Repository: Encapsulates business rules in a reusable manner, integrating seamlessly with the repository for clean and maintainable data access.
6. Hangfire with Redis: Hangfire is integrated for background processing and managing recurring jobs. Redis is used as the job storage backend, offering high performance and reliability.

**Workflow:**

1. **Request Processing:** The ASP.NET Core layer receives and processes requests with minimal overhead.
2. **Command/Query Dispatch:** These requests are transformed into commands or queries, dispatched using MediatR.
3. **Core Processing:** The Core Project executes business logic and interacts with the data access layer.
4. **Data Interaction:** EF Core handles standard data operations, while Dapper manages complex queries.
5. **Background Jobs:** Hangfire schedules and executes background tasks and recurring jobs, with Redis ensuring efficient job storage and management.
6. **Response Handling:** Finally, the web layer sends the appropriate responses to the client.

**Key parts of the codebase:**

| 1 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/docker-compose.yml | We use docker-compose to bring up dependencies of web-api while working locally. Run **docker compose up** in the root of the repository to bring up all the dependencies. |
| --- | --- | --- |
| 2 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Web/appsettings.json | Defines all the configuration required by code to run eg. DB Connection strings, api keys to third party servers etc. The values for cloud environments are hosted on AWS parameter store |
| 3. | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Web/appsettings.Local.json | The local configuration file bootstrapped with default configuration like connection string, api keys etc. |
| 4. | https://github.com/cawstudios/BeTagged.CoreAPIs/tree/develop/BeTagged.Web/ApiControllers | All the controllers for betagged are created here and versioned. All containers are protected (requires authZ) by default. We can use FE to see what API calls are happening on user’s action to get the controller. |
| 5. | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Core/Services/Security/ICurrentUser.cs | To get the information of the user sent the request to backend. We can inject this type in any service (not background). Use this when you’re not sure what type of user we’re dealing with or the code is common for both **Influencer** and **BrandMember** user type. |
| 6. | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Core/Services/Security/ICurrentBrandMember.cs | To get the information about a user (**Brand Member)** sending the request to backend. |
| 7 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Web/Services/CurrentInfluencer.cs | To get the information about a user **(Influencer)** sending the request to backend. |
| 8 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Web/Infrastructure/Filters/AllowedUsersAttribute.cs | To specify what type of user is allowed to access a certain Controller or Action method |
| 9 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Web/Infrastructure/Filters/BtAuthorizeAttribute.cs | To specify the role needed by **BrandMemebr** to access a certain endpoint or API |
| 10 | https://github.com/cawstudios/BeTagged.CoreAPIs/tree/develop/BeTagged.Data/Entities | All Entities (table) are created in this folder. |
| 11 | https://github.com/cawstudios/BeTagged.CoreAPIs/tree/develop/BeTagged.Data/EntityConfigurations | All entity related configuration eg. pk, indexed, fk are defined here. We don’t pollute Entity class with EF core specific stuff. |
| 12 | https://github.com/cawstudios/BeTagged.CoreAPIs/tree/develop/BeTagged.Data/Sqls | Raw SQLs are created in this folder |
| 13 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Data/Enums/BtQueryType.cs | When you create a SQL file. Then, make to add it’s entry to this enum |
| 14 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Data/Services/IQueryService.cs | Inject this service to execute Raw SQLs and to get the result on CLR classes |
| 15 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Data/Repositories/IRepository.cs | Inject this file to call datbase using EF core. This is a generic class and entity needs to defined as generic type. The methods of this files requires specifications which will be executed by the implementation of this interface. |
| 15 | https://github.com/cawstudios/BeTagged.CoreAPIs/blob/develop/BeTagged.Core/Specifications/Queries/GetUserKeySpecification.cs | Example file to showcase. How, we’re usign specification pattern with EF Core. |

**Getting started:**

1. Run **docker compose up** in the root of the cloned repository to bring up the dependency.
2. Create a **.env** file <https://github.com/cawstudios/BeTagged.CoreAPIs/tree/develop/BeTagged.Web> folder and make sure you’ve following values configured.

AwsConfig\_\_AccessKeyId=**<YOUR\_AWS\_ACCOUNT\_ACCESS\_KEY\_ID>**

AwsConfig\_\_AccessKeySecret=**<YOUR\_AWS\_ACCOUNT\_ACCESS\_KEY\_ID>**

AwsConfig\_\_S3BucketName=betagged-dev-bucket

AwsConfig\_\_S3CloudFrontDomain=https://storage-dev.betagged.co/

AwsConfig\_\_S3CloudFrontUrlSignerKeyPairId=K1V0ALFAYR5Q7I

AwsConfig\_\_S3CloudFrontUrlSignerPrivateKey=**<GET\_FROM\_DEV\_AWS\_PARAMETER\_STORE>**

# GeoDbApiConfig

GeoDbApiConfig\_\_ApiKey=**<GET\_FROM\_DEV\_AWS\_PARAMETER\_STORE>**

SendinBlue\_\_ApiKey=**<GET\_FROM\_DEV\_AWS\_PARAMETER\_STORE>**

# shopify

ShopifyConfiguration\_\_Url=http://betagged-incom.myshopify.com/

ShopifyConfiguration\_\_AccessToken=**<GET\_FROM\_DEV\_AWS\_PARAMETER\_STORE>**

1. Run the **BeTagged.Web** project. You should be greeted with **Swagger** page.

### 

### 

### 

### [BeTagged.UrlShortener](https://github.com/cawstudios/BeTagged.UrlShortener)

It is a custom URL Shortener.

Framework: DotNet Core WebAPIs

**Version: 6**

**Files: 133**

**KLOC:76349**

**Repo:** <https://github.com/cawstudios/BeTagged.UrlShortener>

**Overview of components:**

**1. UrlShortener.Web**

This is the frontend and API layer of the system. It serves two main purposes:

1. **URL Shortening:** It provides an interface for users to shorten URLs. When a user inputs a long URL, this module generates a unique short URL.
2. **URL Redirection:** When someone clicks on a shortened URL, the request is routed to this project. The corresponding controller is responsible for handling the redirection to the original URL.

**Key Features:**

1. **API Endpoints:** RESTful endpoints for URL shortening. <https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/ApiControllers/ShortenedUrlApiController.cs>
2. **Redirection Logic**: A controller to handle the redirection logic based on the short URL identifier. <https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/Controllers/HomeController.cs>

**Command Queueing:** Upon URL redirection, it queues a command in an AWS SQS queue to increment the click count.

**2. UrlShortener.Lambda**

This component is an AWS Lambda function that gets triggered by events in the AWS SQS queue.

**Key Functions:**

1. **Click Count Management:** Once triggered, it increases the click count of the corresponding URL.
2. **Webhook Notification:** After updating the click count, it sends a webhook notification to the client to inform them of the click event.

**Inter-component Communication**

When a short URL is accessed, UrlShortener.Web queues a command in the SQL queue.

The SQL queue triggers UrlShortener.Lambda.

UrlShortener.Lambda processes the event, updates click counts, and sends a webhook to the client.

**Technologies and Patterns:**

**.NET Core:** The entire solution is built on .NET Core, offering cross-platform compatibility and performance benefits.

**SQS Queue:** Used for decoupling UrlShortener.Web and UrlShortener.Lambda, ensuring scalability and reliability.

**AWS Lambda:** Provides a serverless architecture, reducing the need for server management and scaling automatically with the number of requests.

Webhooks: For asynchronous communication back to the client about click events.

**Key components of code**

| 1 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/appsettings.json | Contains all the config required by the project to run. These values are hosted on AWS parameter store |
| --- | --- | --- |
| 2 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/appsettings.Local.json | Bootstrap config for local environment |
| 3 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/URLShortener.Lambda/appsettings.json | Appsettings for AWS Lambda project |
| 4 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/docker-compose.yml | Compose file containing all the dependencies to run the web project in local |
| 5 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/Startup.cs | All the services required by the code eg SQS, Dynamo DB etc are configured here. |
| 6 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/ApiControllers/ShortenedUrlApiController.cs | This controller is used by the client to create a short url. The controller is protected. |
| 7 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/Infrastructure/Filters/ApplicationClientAuthenticationFilter.cs | This filter is used to validate the client id and client secreted for above controller |
| 8 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/UrlShortener.Web/Controllers/HomeController.cs | This controller gets called when the short url is clicked and push and event to SQS |
| 9 | https://github.com/cawstudios/BeTagged.UrlShortener/blob/develop/URLShortener.Lambda/ClickEventSqsLambda.cs | This AWS lambda event handler is triggered by above SQL event. This increment the click count of the short url entry in the database and send a webhook event to client owning the short url i.e. **BeTagged.Web** |

**Data Storages**

* PostgreSQL as our primary DB
* Dyanmo db to store the **Client** and **ShortenedUrls**
* Redis for Hangfire Jobs
* S3 for Images and storage for PO docs