Group 15 Documentation

User manual:

Testers:

Testers need to have a username and password created for them (see authorization). Once logged in, testers will automatically be redirected to their dashboard which contains a button to enter the results of a new positive case. Testers are expected to add all cases with the form provided. Required fields are marked, if the postcode of a patient is not known the field can be left blank to default to that of the testing centre.

Tracers:

Tracers need to have a username and password created for them (see authorization). Once logged in, tracers will automatically be redirected to their dashboard. Here a tracer should see all of the cases assigned to them for tracing. Clicking on a case brings up the name and contact information for that case. A tracer is expected to get in contact with the case and ask them who they have been in close contact with (as per up to date definitions of “close contact”). A tracer can record the details of these contacts using the buttons and form provided. If an email address for a contact has not been provided, a tracer is expected to phone the contact themselves to give them up to date information regarding their exposure and what to do going forwards. Where an email address is given, a tracer does not need to personally get in contact with them. If a tracer is unable to contact a case, a button is provided to drop them. This will either assign the case to another tracer for another attempt or permanently un-assign the case from all tracers as a lost cause.

Government actors:

Government actors are not logged in via our database, and only need a correctly configured Okta account to log in (i.e. account type = “2”). A logged in user is met by the unpersonalised dashboard, including six key summary statistics and three buttons. The leftmost button brings up a heatmap of cases in a given date range (that they can alter), intended to be used to generate graphics for non-specialist consumption. The central button brings up a new page with a table of contact tracing centres. Each centre has statistics on its performance in the last 28 days listed in its row, including average time between case creation and submitting a case, the number of cases assigned and reached, and the percentage of cases reached. The rightmost button triggers a file download of a spreadsheet. This spreadsheet consists of all fields in all tables stored in the database that aren’t personal data, and is intended to facilitate further data analysis.

Technical documentation:

This section focuses on documenting areas of our project that aren't present in the code or are otherwise critical to understanding how the solution functions.

Initialisation:

Running our system begins with *Program.cs*, a standard asp.NET file that creates a host and runs it. As part of creating this host, the methods contained in *Startup.cs* are run, in particular *ConfigureServices* and *Configure*. *Configure* is used to set up the use of authentication, authorization and https, as well as specifying settings for production vs development scenarios.

ConfigureServices is where any services required by the page classes are instantiated, and dependency injection is set up. For example, the tester form page needs to be able to access the database to enter a new positive case. Access to the Case table is managed by the *SQLCaseRepository* class, so to avoid creating multiple instances of this class (a potential concurrency nightmare) an *SQLCaseRepository* service is logged in *ConfigureServices.* Then, whenever an *SQLCaseRepository* instance is required, asp.NET's dependency injection system will provide an actual instance. This process is required for all of our services (all of the classes listed in *Services* except the database migrations).

Once these two classes have done their work, the hosting service being used (IIS locally or Azure when deployed) handles requests via multi-threading.

User secrets:

Besides the connection string to the database, there are only two secret values required by the project.

1. "googleApiKey": Used to access Google's map API for running the heatmap and postcode geocoding (converting postcode to latitude and longitude).
2. "emailPassword": The password to the email address used to automatically send email messages to confirmed contacts.

When running locally, these values should be set using the secrets.json file. When running deployed, these can be configured via the Azure portal (Configuration tab) as "application settings". These values are used by the code via the dependency injection of an *IConfiguration* object.

Authentication with Okta:

Okta, as a third party authentication system, helps our team outsource the fine details of authentication works and thus accelerate our development. The service on Okta's end responsible for login is "call an application".

Okta communicates with our project mainly through URIs. Our website requests a log in through Initial Login URI, and receives back the details of the user through Login Redirect URI. Log out is set up in the same way. These are set up on https://developer.okta.com/.

Secured connection between our website and Okta through client id, client secret and client domains. These configurations are stored in Startup.cs. The control logics are managed by AccountController, with corresponding front end in login_partial.cshtml shared by all webpages

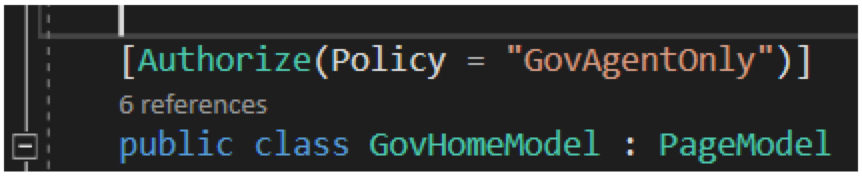
Currently all the users are created on Okta then added into the application manually, mainly for security reasons.

The current users associated with the application are:

Tester2@123.com dummypassword
Tracer2@123.com dummypassword
Tracer3@123.com dummypassword
Gov2@123.com dummypassword

Authorisation:

Authorisation policy tags are used to ensure only correctly authenticated users are able to view pages. The tag can be placed above any PageModel class to restrict access to the page, see image.



```
[Authorize(Policy = "GovAgentOnly")]
6 references
public class GovHomeModel : PageModel
```

Three policies are implemented, one for each type of user (Gov agents, Testers and Tracers). These policies are instantiated in the *Startup.cs* file, and make use of the *UserTypeRequirement* class to check the currently logged in user's role identity, which are stored as claims. Testers claim a usertype of "0", tracers claim "1" and government agents claim "2".

Database tables:

We instantiate six database tables: TracingCentres, TestingCentres, Tracers, Testers, Cases and Contacts. Tracers belong to a tracing center, testers belong to a testing centre. A case belongs to one tester and one tracer, while a contact has a tracer and a case associated with it. A database that contains usernames for one tester and two tracers (together with their associated centres) will be instantiated upon running the program for the first time.

Database access:

Database connection and data management are done by Entity Framework with some SQL queries. All of database abstraction is contained in Models and Services projects.

Models:
Contains code-first implementations of database Rows. With entity framework, individual rows are treated as objects in the code with Navigational Properties standing for Foreign Keys.

Services:
Contains code-first creation of the database and it's abstraction from the code.

Migrations folder contains migrations - code-first commands to create the database. Most of the migrations are automatically generated from the Models and AppDbContext.

Repositories folder contains repositories - first level of abstraction that contains implementation of all the queries, which are a combination of Entity Framework fluent API and raw SQL.

Repository Interfaces contain functions used directly by the program. They mostly use Repository methods with some extra Fluent API.

AppDbContext contains the code-first model of the database, containing the Model - tables of the database and function ran on database creation (in our case - seeding). This file is used to generate the migrations together with Models.

AppDbContextFactory creates a concrete instance of AppDbContext with SQL server connection

Deployment:

The two main technologies used to deploy the web application are Docker and Azure. Docker is a service that bundles dependencies and code into a "container", which can then be run without external dependencies. Azure is Microsoft's web hosting service, which we are using with an "Azure for students" account for a limited amount of free hosting.

The top-level file "Dockerfile" is used to instruct Docker in how to successfully convert the application into a runnable container. Then the app is published with the press of a button in Visual Studio, by creating the container, and placing it in an Azure Container Registry. This updated container is then taken from the registry by our Azure App Service instance and runs on a web server.

The online database used is also hosted on Azure, as an Azure SQL Database instance. Azure manages the dependency with its Secret Store used to configure and store the connection string to the database server. The *AppDbContext* used throughout the codebase is automatically configured to connect to the online database when the solution is deployed online, and to connect locally when run locally.